BRL R 1346

BRL

AD

REPORT NO. 1346

BRLESC FORTRAN 1V

by

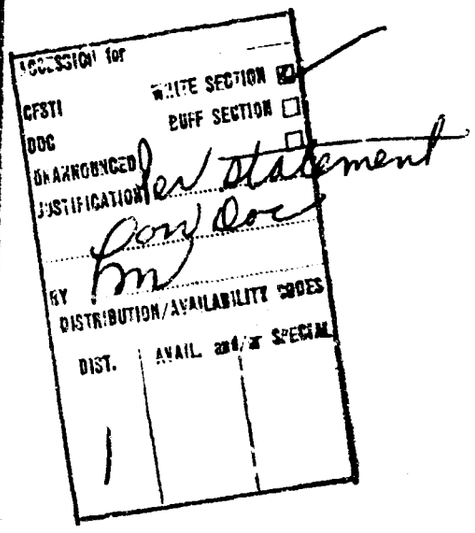
Lloyd W. Campbell
Glenn A. Beck

October 1966

Distribution of this document is unlimited.

U. S. ARMY MATERIEL COMMAND
BALLISTIC RESEARCH LABORATORIES
ABERDEEN PROVING GROUND, MARYLAND

Destroy this report when it is no longer needed.
Do not return it to the originator.



The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

BALLISTIC RESEARCH LABORATORIES

REPORT NO. 1346

OCTOBER 1966

BRLESC FORTRAN IV

Lloyd W. Campbell
Glenn A. Beck

Computing Laboratory

RDT & E Project No. 1P014501A14B

ABERDEEN PROVING GROUND, MARYLAND

B A L L I S T I C R E S E A R C H L A B O R A T O R I E S

REPORT NO. 1346

LWCampbell/GBeck/saf
Aberdeen Proving Ground, Md.
October 1966

BRLESC FORTRAN IV

ABSTRACT

FORTRAN is a popular programming language that has been implemented on many computers. It is now available on Ballistic Research Laboratories' BRLESC computer. This report describes the FORTRAN language in general and includes specific details about its implementation on BRLESC.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

I. INTRODUCTION

FORTRAN is a programming language that is widely used on a variety of computers and can be used on Ballistic Research Laboratories' BRLESC computer. FORTRAN was designed primarily for programming of scientific problems and the evaluation of arithmetic formulas. It is basically similar to the FORAST programming language that is currently used on BRLESC but many of the details are different.

This manual is intended primarily for the programmers that are familiar with FORAST and BRLESC; however, it includes a general description of the FORTRAN language and should prove helpful to anyone who is interested in writing or reading FORTRAN programs. Additional details and general information can be obtained from other FORTRAN manuals and publications. The FORTRAN IV manuals for the 7090/7094 are suggested for those interested in using BRLESC FORTRAN since BRLESC FORTRAN is more compatible with the 7090/7094 version of FORTRAN than with some versions that are used on other computers. Some readers may be surprised to learn that FORTRAN is not the same for all computers. Although the general rules are usually the same, differences in details do exist and some of these differences are quite subtle. It is relatively easy to write FORTRAN programs which when executed on different computers will yield different results. These differences may be due to differences in compilers or differences in the structures of the computers. However, most FORTRAN programs require only minor modifications to allow them to run on any given computer. The modifications usually require much less effort and time than would be required to re-program the problem in another programming language.

There have been two prominent versions of the FORTRAN language. They are referred to as FORTRAN II and FORTRAN IV. FORTRAN IV does not include everything that was in FORTRAN II. However, the BRLESC FORTRAN IV compiler has retained essentially all of FORTRAN II so that it will accept statements that are defined in either of these two versions of the FORTRAN language.

II. THE CHARACTER SET

FORTRAN allows the use of the twenty-six capital letters of the alphabet, the decimal digits 0 to 9 and the special symbols + - ( ) . * / , = $.

The card code for these characters is the same as normally used for BRLESC and other computers except for the following: (These exceptions will be removed in January 1967.)

FORTRAN BRL

(% Standard FORTRAN left parenthesis uses 0-4-8 punches that represents % at BRL. Since FORTRAN does not allow the 4-8 code (BRL left parenthesis) in programs, BRLESC FORTRAN allows either code to mean left parenthesis. (After January 1967, BRL will use the standard 0-4-8 punches for left parenthesis and the 4-8 code will not be allowed.)

\+ -
\- + Standard FORTRAN uses card codes for the signs that are just the opposite of present BRL usage (+ is x, - is y at BRL). (BRL will change to standard FORTRAN signs in January 1967.) A "CHANGE + AND -" control card may be inserted in a FORTRAN program to cause BRLESC to reverse these symbols. They are initially set for BRL usage.

The 709/7090 FORTRAN and BRLESC FORTRAN also allow a 4-8 card code to be a minus sign on <u>decimal input</u>. BRL signs are used for decimal input unless the SETMSI subroutine is used to change signs. A "CHANGE + AND-" control card <u>does not</u> change signs used for input data.

$ ' This card code X-3-8 is allowed only in hollerith text (H fields) in FORMAT statements. (After January 1967, the $ character will be used as an end of statement mark, instead of > , and will also replace the % character on BRLESC assembly cards, both in column one and between orders.)

III. SYMBOLIC NAMES AND CONSTANTS

General Names

In FORTRAN, all symbolic names (other than statement names) must begin with a letter and, for variables, the first letter usually determines the type of number it represents. Names of arithmetic variables that begin with I, J, K, L, M or N represent integer numbers unless they are declared to be floating point in a REAL statement. Names beginning with other letters represent floating point numbers unless they are declared to be integers in an INTEGER statement. Names of logical variables may begin with any letter and must be declared in a LOGICAL statement.

The length of symbolic names is restricted to six characters except subroutine names may have a terminal F as a seventh character. If names longer than six characters are used on BRLESC, the first five characters and the last non-F character will be used as the name. Arithmetic statement function names must not have more than seven characters.

Statement Numbers

Locations of statements (cols. 1-5 of FORTRAN statement cards) must be all decimal digits and thus look like integer numbers but are really symbolic locations of statements. Leading zeros and blank columns are ignored. (Statement numbers must be less than 32768 for 7090/7094 but not BRLESC.) They do not affect the sequence of execution of the statements. On BRLESC, statement numbers may be written in place of a variable name by writing an S after the statement number.

Constants

1. Integer constants are written without a decimal point. An integer constant on BRLESC may consist of 1 to 17 decimal digits. Some computers restrict integer constants to as few as four decimal digits. The values of integer variables on BRLESC must be less than 2^{64} in absolute value except the divisor and quotient of integer divide operations must both be less than 2^{34} in absolute value.

2. Floating point constants must be written with a decimal point. They may consist of a decimal point with 1 to 17 decimal digits (on BRLESC) and may be followed by an E and a decimal exponent. (BRLESC also allows a D instead of E to indicate an exponent. The D indicates double precision constants on most other computers.) The BRLESC range of floating point constants (and variables) is between 10^{155} and 10^{-155} approximately in absolute value with zero also allowed. Most computers have a more restricted range of numbers.

Examples: 1. , 4.21 , .2 , 51.6 E2 , .1E-3 , 3.1 D-1

3. Alphanumeric constants of six or less characters are allowed on BRLESC. They must be preceded by nH where n is the number of characters in the constant. Blanks are not ignored in the n columns after the H.

4. The logical constants allowed in FORTRAN IV are ".TRUE." and ".FALSE.". Note the use of a period at both the beginning and end of these constants.

5. Octal constants are written as twelve octal digits. If less than twelve digits are written, zeros are added by the computer to the left of the digits to make a total of twelve. Octal constants are only allowed on FORTRAN II type boolean cards with a B in column one and in DATA statements. In DATA statements only, the octal digits must be preceded by the letter O.

Arrays

Blocks of storage are referred to as arrays in FORTRAN and are defined in DIMENSION, COMMON or TYPE statements. Subscripts are enclosed in parentheses in FORTRAN, e.g. A(3) or B(I,J), and one, two, or three dimensional arrays may be used. Any subscript may be variable and "indexing", as done in FORAST, is not allowed.

Subscription of variables is done by substitution rather than addition and the lower bound of all subscripts is one. Subscript arithmetic is allowed; BRLESC FORTRAN allows any integer arithmetic expression that does not itself involve any subscripted variables, however, the most general expression allowed in standard FORTRAN is C * V $\pm$ C' where C and C' are integer constants and V is an integer variable.

Symmetric arrays are not allowed in FORTRAN and there is no provision for "interweaving" arrays.

Absolute Addresses

Absolute decimal or sexadecimal addresses are allowed only in BRLESC assembly language instructions.

IV. ARITHMETIC EXPRESSIONS

The following symbols denote the following operations:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

The use of functions (subroutines with only one result) is also allowed by writing the name of the function in front of parentheses that enclose the arguments. (FORTRAN allows functions to have more than one argument and commas are used to separate the arguments.) The arguments may be arithmetic expressions.

The precedence of operations when not governed by the use of parentheses is

functions (subroutines)
**
* and /
+ and -

where the operations higher on the list will be performed before those

that are lower on the list. Successive + and - operations or successive * and / operations will be performed from the left to the right. Parentheses may always be used to cause the operations to be done in any desired sequence. Successive exponentiations must always have parentheses to show the desired grouping.

The major difference between FORAST and FORTRAN arithmetic expressions is the grouping of successive multiplications and divisions. FORAST groups them from the right and FORTRAN groups them from the left. Thus in the expression (A * B/C * D), D is in the denominator for FORAST and is part of the numerator for FORTRAN.

Implied multiplication should not be used in FORTRAN (although some versions do allow it and BRLESC FORTRAN allows it after a right parenthesis).

Fixed point fractional arithmetic is not allowed in FORTRAN. All arithmetic within an expression must be one mode (integer or fl.pt.) except for integer subscripts and integer powers of exponentiation in floating point expressions. The first letter of the names and the use of the decimal point in numbers determines the mode rather than preceding the expression with a declaration of the mode as is done in FORAST.

Parentheses must not be omitted at the ends of an expression. The number of left parentheses must be the same as the number of right parentheses in each expression.

Two operations must not appear adjacent to each other in formulas; e.g.,/ - or ** - .

Any operation on integers which does not yield an exact integer result is truncated except negative integer results of division on BRLESC FORTRAN will give the greatest integer that does not exceed the algebraic exact result. Thus -4.2 will give -5. This is probably different from results on the 7094 or other absolute value machines.

From FORTRAN II on the 7090/7094, boolean expressions are allowed on cards with a B in column one. The symbols +, * , - denote the logical operations of or (inclusive), and, and complement respectively. BRLESC FORTRAN performs these operations only on the rightmost 36 bits of a word so that it is compatible with the 36 bit word length of the 7090/7094. The leading 32 bits of a BRLESC word will be zeros after a logical operation. Note that FORTRAN IV has provided a new way of writing these logical operations as explained in Section VI below.

Double Precision arithmetic expressions are allowed (a D in col. 1) but are done in BRLESC single precision which is as accurate as 7090/7094 double precision.

Complex arithmetic expressions are not presently allowed in BRLESC FORTRAN. An I in column one will cause an error print.

V. ARITHMETIC FORMULAS

The general form of FORTRAN arithmetic formulas (arithmetic statements) is

v = ae

where v is a name of an arithmetic variable (it may be subscripted) and ae is an arithmetic expression. An example would be

X(J + 1) = A(J)**2 - V/(T + 3.)

The arithmetic expression is evaluated and the result is stored as the new value of the variable whose name is on the left of the = symbol.

No arithmetic may be performed on the left of the = symbol except for subscript arithmetic. Only one = symbol is allowed and hence only one variable will have its value changed by an arithmetic formula. If the type of this variable is different than the type

of the expression on the right of the = symbol, the value of the expression is automatically converted to agree with the type of the variable on the left of the = symbol before it is stored.

The arithmetic expression may be just a name of a variable or constant, e.g.,X = A.

VI. LOGICAL EXPRESSIONS

FORTRAN IV permits the use of logical variables and expressions that assume either the value .TRUE. or the value .FALSE.. The following three logical operations are defined using a and b to represent logical variables or logical expressions:

.NOT.a	is .TRUE. when a is .FALSE. and is .FALSE. when a is .TRUE.
a.AND.b	is .TRUE. when both a and b are .TRUE. and is .FALSE. when either a or b or both are .FALSE.
a.OR.b	is .TRUE. when either a or b or both are .TRUE. and is .FALSE. only when both a and b are .FALSE.

Two adjacent logical operations may be used only when the second one is .NOT.. Thus .AND..NOT. is legal but .NOT..AND. is illegal.

A relational expression that consists of a comparison of two arithmetic variables or expressions may be used to form logical expressions. FORTRAN IV uses the following relational operators: (x and y represent arithmetic variables or arithmetic expressions.)

x.EQ.y	is .TRUE. only if $x = y$.
x.NE.y	is .TRUE. only if $x \neq y$.
x.GT.y	is .TRUE. only if $x > y$.
x.GE.y	is .TRUE. only if $x \geq y$.
x.LT.y	is .TRUE. only if $x < y$.
x.LE.y	is .TRUE. only if $x \leq y$.

Whenever the relational expression is not .TRUE., it is .FALSE.. The arithmetic quantities x and y must be of the same type in any one relation, e.g.,if I is integer in I.LT.J, then J must also be integer.

On BRLESC, the operands for .EQ. and .NE. could be logical variables but this is not true for most other computers.

It is illegal to use one arithmetic quantity as the operand for more than one relation. Hence the mathematical expression $x < y < z$ must be written as X.LT.Y.AND.Y.LT.Z and not as X.LT.Y.LT.Z.

A logical expression is any legal combination of logical operations and relational expressions. Parentheses may be used to obtain any desired grouping of operations. In the absence of parentheses, the operations are performed in the following order:

Arithmetic operations; Functions	
	**
	* and /
	+ and -
Relations; .LT..LE..EQ..NE..GT..GE.	
Logical operations;	.NOT.
	.AND.
	.OR.

Note that all the relations have equal precedence which means that they will normally be evaluated from left to right. Note also that .NOT. has a higher precedence than .AND. and .OR. and hence will be performed before the other two logical operations.

Logical Assignment Statements:

Logical expressions may be used in logical IF statements (see Section VIII, item 6) and in logical assignment statements. Logical assignment statements have the general form

a = le

where a is the name of a logical variable and le is a logical expression. The value stored in a will be .TRUE. or .FALSE. as determined by the evaluation of the logical expression le.

Examples of logical assignment statements:

(I,J,X and Y represent arithmetic variables and A,B and C represent logical variables.)

```
A = .FALSE.
C = A.AND..NOT.B
B = .NOT.(A.OR.B)
A = I.LE.3
B = I.EQ.J.AND.(B.OR.X.LE.Y)
C = 3.1416.GT.X+Y.OR.I*J.GT.1000
```

Logical Masking Statements:

To improve compatibility with CDC FORTRAN, BRLESC allows logical masking statements. The operations .NOT.,AND., and .OR. may be used with arithmetic operands to accomplish bit-by-bit logical operations using the last 36 bits of BRLESC words.

An example of a logical masking statement would be

```
T = X.AND..NOT.Y
```

where X and Y are arithmetic variables (real or integer) and T may be any type of variable. This example will do a bit-by-bit product of X and the complement of Y and will store this result in T without any conversion.

VII. SPECIFICATION STATEMENTS

This group of statements (DIMENSION, COMMON, EQUIVALENCE, TYPE, EXTERNAL and FREQUENCY) provides information to the compiler and may be used by the programmer to control the storage assignment of some or all of the variables. These statements do not cause any machine code to be generated for running the program; they only affect the way it is compiled.

DIMENSION a(i), b(i1,i2), c(i3,i4,i5),..., where a,b,c are array names and the i's are integer constants.

This statement is used to declare the names and maximum sizes of arrays. The maximum subscripts are enclosed in parentheses and they must be decimal integer constants except integer dummy variables may be used in FORTRAN IV subprograms if the array being defined is also a dummy variable. (See SUBROUTINE statement description.) The minimum subscript is always taken to be one. One, two, or three dimensional arrays may be defined in any sequence.

An array must be declared before its name is used in any other statement. FORTRAN IV allows arrays to be declared in DIMENSION, COMMON or TYPE statements with only one declaration allowed for the same array.

Example: DIMENSION T(41),X(10),E(4,4,4),A(3,7)

COMMON a,b,c,d,e ..... , where a,b,c,d,e are the names of variables of any type.

This statement allows the programmer to specify that certain variables and arrays are the same in more than one program or subprogram (subroutine or function). The storage assigned to those items in the COMMON statement in one subprogram is the same storage assigned to the items in the COMMON statements in all of the other subprograms (and also the main program). Thus it also has an equivalence effect between subprograms. All storage used in each subprogram is different than the storage in any other subprogram except for the items that are listed in COMMON statements.

Within each subprogram, all COMMON variables are assigned consecutively in the sequence in which they appear. The starting point for all the subprograms within each total program is the same. Proper space is left for arrays.

COMMON statements are used to avoid listing many arguments when using a subprogram. By forcing the main program and subprograms to use the same storage for some (and possibly all) of the variables, the need for specifying and moving variables is removed.

If any COMMON variable also appears in an EQUIVALENCE statement, the COMMON assigning has priority and is done first in BRLESC FORTRAN. This is different from 7090/7094 FORTRAN II where EQUIVALENCE variables are assigned first and will change the sequence of storage assigned to COMMON variables. BRLESC FORTRAN handles the COMMON-EQUIVALENCE interaction as specified for FORTRAN IV.

FORTRAN IV allows dimension information to be specified in COMMON statements. However any one array must not be dimensioned more than once in the same program or subprogram, i.e.,if an array name in a COMMON statement contains dimension information, it must not also be dimensioned in a DIMENSION or TYPE statement.

FORTRAN IV allows labeled COMMON blocks. A group of names may be preceded by a slash, a label name and another slash to give a name (label) to a section of the COMMON storage area. By using labeled COMMON, it is no longer necessary to think of COMMON as one big block. Whenever the same label is used in different subprograms, the corresponding members of the two labeled blocks will be assigned the same storage positions regardless of their relative position within their respective COMMON statements. The following example will illustrate the meaning of labeled COMMON. If the following COMMON statements each appear in a different subprogram within the same complete program,

```
COMMON A,X/LA/B,I,W/AA/P,M,N
.....
COMMON A,Y/AA/P,M1,N1/LA/E,J,W//Z
```

then the names A,P and W refer to the same quantity in both of the subprograms. The names X,B,I,M and N within the first subprogram refer to the same quantities respectively as the names Y,E,J,M1 and N1 in the second subprogram. In the second subprogram, the blank COMMON consists of A,Y and Z because two consecutive slashes

cause the following quantities to be added to the blank COMMON block. Blank COMMON blocks do not have to be the same length in each subprogram. However labeled COMMON blocks of the same label must be the same length whenever they are used in different subprograms within the same complete program. (Length is defined as the amount of memory space used.) Label names may be any legal FORTRAN name except the names of subroutines and functions may not be used. It is permissible to use the same name for a label and a variable within the same subprogram.

A subprogram may have more than one COMMON statement. Additional COMMON statements simply extend the list of COMMON variables. The use of the same label again within the same subprogram simply extends the list of variables in that labeled block. Thus the two consecutive statements

```
COMMON A,B,C/T/F,G
COMMON E/T/R,S//V
```

is the same as the single statement

```
COMMON A,B,C,E,V/T/F,G,R,S
```

On BRLESC, the statements COMMON(USE MAIN) or COMMON(USE PREVIOUS) may be used instead of repeating long COMMON statements in a subprogram when all of the COMMON variables are identical with the main program or the previous subprogram.

EQUIVALENCE (a,b,c,...),(d,e,f,...), where a,b,c,d,e,f are names of any type of variable.

This statement causes different names to be assigned to the same memory space. (It performs the same function as SYN does in FORAST.) All the names within a set of parentheses are made equivalent. Increments may be used if desired by enclosing them in

parentheses immediately after the name. (No increment is the same as an increment of one.) Increments on array names may be either a single integer constant or FORTRAN IV allows the proper number of integer constant subscripts to be used.

Whenever arrays are partially or completely overlapped, space is always reserved for all of the arrays involved so that there is no unexpected overlapping of storage. However, EQUIVALENCE will not re-arrange COMMON storage; so equivalencing a larger array with a member of COMMON may cause additional overlapping of storage space.

It is illegal to use EQUIVALENCE to try to cause any impossible arrangement of storage. It cannot be used to attempt to cause non-consecutive spaces to be assigned to elements of an array, to extend the beginning of the COMMON storage area or to equivalence two variables that are both in COMMON. It is also illegal for names of dummy variables to appear in an EQUIVALENCE statement.

On BRLESC, it is illegal to equivalence anything to itself, either directly or indirectly.

On BRLESC, any EQUIVALENCE statement that contains the names of arrays and variables that are in COMMON statements must appear after the DIMENSION and COMMON statements.

Example: EQUIVALENCE (A,B),(F(2,1),C,H(1))

TYPE STATEMENTS

FORTRAN IV allows TYPE statements that declare specified variable names to represent variables of a specified type. If a name does not appear in a TYPE statement, then its first letter determines whether it represents an integer or a real (floating point) number. However, a TYPE statement near the beginning of a program may be used to overide the automatic type assignment.

BRLESC allows the following TYPE statements:

```
INTEGER a,b,c,...
REAL a,b,c,...
DOUBLE PRECISION a,b,c,....
LOGICAL a,b,c,...
```

where a,b,c,... represents a list of variable and function names. On BRLESC, DOUBLE PRECISION is used the same as REAL since double precision on most other computers is the same as BRLESC's single precision.

Some computers also allow a COMPLEX statement but this causes an error print on BRLESC because complex arithmetic is not allowed.

Variable array names in TYPE statements may also contain dimension information. However the same variable must not also be dimensioned elsewhere, i.e.,it must not also appear in a DIMENSION statement or be dimensioned in a COMMON statement.

The names of all logical variables must be declared in a LOGICAL statement as there is no other method of distinguishing them from other variables.

The TYPE statements must precede any executable statement or DATA statement that uses any variable or function mentioned in a TYPE statement. Note that these TYPE statements are non-executable; they cannot be used between executable statements to cause any run-time data conversion.

BRLESC FORTRAN IV allows any of these type statements to be preceded with the word TYPE because CDC FORTRAN allows this. It is for this reason that the names TYPEI, TYPER, TYPED, TYPEL and TYPEC must not be used as names of variables at the beginning of any statement. (CDC and BRLESC do not use the word PRECISION when DOUBLE is preceded by TYPE.)

Examples of some TYPE statements:

```
REAL MASS, N2,LA(5,6),X
INTEGER A,F,I(15)
LOGICAL LV,T,WAY,LOW(18),NOW
TYPE REAL M1,M2
```

EXTERNAL STATEMENT

FORTRAN IV allows this statement to be used to specify the names of subroutines and functions that are used as arguments for other subroutines or functions. It serves the same purpose as the card with F in column one did in some FORTRAN II compilers. The general form of the statement is

```
EXTERNAL a,b,c,...
```

where a,b,c,... represents a list of function and subroutine names.

For BRLESC, any arithmetic statement function names used as arguments must also appear in an EXTERNAL statement.

If the name of a function appears in both a TYPE statement and an EXTERNAL statement, the TYPE statement must precede the EXTERNAL statement.

Example:

```
EXTERNAL SIN,COS,FUN
```

FREQUENCY

This statement is ignored by BRLESC FORTRAN. Its purpose in 7090/7094 FORTRAN is to provide information that helps the compiler to optimize the program.

VIII. CONTROL STATEMENTS

This group of statements provides for controlling the sequence in which statements are executed in the running program. Unconditional transfer of control (sometimes called branching or jumping) is provided for by several types of GOTO statements and conditional transfer of control is provided by several types of IF statements. A DO statement allows definition of a "loop" and a CALL statement causes transfer of control to a subroutine with a return to the next statement. There are two statements (STOP and PAUSE) that cause the program to stop running.

1. GOTO s where s is a statement number.

This statement causes the statement numbered s to be done next.

Example: GOTO 22

2. GOTO (s1,s2,s3,...),i where s1,s2,s3 are statement numbers and i is a nonsubscripted integer variable.

This is referred to as a "computed GOTO" and the statement done next depends on the value of i. If i = 1, s1 is done next; if i = 2, s2 is done next; etc.

Example: GOTO(4,19,462),K

3. ASSIGN s TO i where s is a statement number and i is a nonsubscripted integer variable.

This statement causes the address of the statement numbered s to be put into the integer variable i and this type of statement is to be executed before the "assigned GOTO" statement (as explained in the next paragraph) is executed.

Example: ASSIGN 64 to M

4. GOTO i, (s1,s2,s3,...) where i is a nonsubscripted integer variable and s1,s2,s3, are statement numbers.

This statement transfers control to the statement that has the number that was last assigned to i by means of an ASSIGN statement. The (s1,s2,s3...) enumeration in this statement is not really

necessary but should be used to list the possible statement numbers that this "assigned GOTO" statement may transfer control to.

```
Examples:  ASSIGN 44 to N
           GOTO N, (16,29,44,192)
```

5. IF (ae) s1,s2,s3 where ae is an arithmetic expression and s1,s2,s3 are statement numbers.

This statement causes control to be transferred to statement s1,s2, or s3 depending on whether the value of the arithmetic expression ae is negative, zero (exactly), or positive respectively.

```
Examples:  IF(X)4,7,22
           IF(R * V - 4.1*(U+V))16,244,16
```

6. IF (le)st where le is any logical expression and st is any executable FORTRAN statement except IF and DO. The statement st is done if the value of the logical expression is .TRUE. and control simply goes to the next sequential statement if the value is .FALSE..

```
Examples:  IF(L.AND.X.GE.Y)GOTO49
           IF(X.LT.10.)X=X+.5
           IF(I+J.EQ.14.OR.PRT)WRITE(2,16)A,B,C
```

7. IF (e) s1,s2 where e is either a logical or arithmetic expression and s1 and s2 are statement numbers. This statement is not standard FORTRAN but is allowed on BRLESC and CDC computers. Statement s1 is done next if e is .TRUE. (or non-zero for arithmetic expressions) and statement s2 is done next if e is .FALSE. (or zero for arithmetic expressions).

```
Examples:  IF(X)22,471
           IF(X.GT.0.AND.L)962,1075
```

8. DO s i = i1, i2, i3 where s is a statement number, i is a nonsubscripted integer variable and i1,i2,i3 are positive integer constants or nonsubscripted integer variables.

This statement causes the statements following this DO statement down to and including the statement numbered s, to be executed repeatedly with the integer variable i initially assuming the value of i1. The variable i is incremented by i3 at the end of the sequence of statements and the sequence is repeated if the value of i does not exceed i2. If i1 > i2 initially, CDC FORTRAN will not execute the loop even once whereas BRLESC and most other computers will always execute a DO loop at least once.

The specification of i3 is optional. If i3 is not specified, its value is taken as one (1).

A DO sequence of statements may itself contain a DO sequence provided the entire inner DO sequence is contained in the outer one. Several DO's may terminate on the same statement.

While most versions of FORTRAN have several other restrictions concerning DO loops, BRLESC FORTRAN does not have any other restrictions on the construction of the statements that are included in DO loops. BRLESC FORTRAN does always set and use the actual integer variable specified for i in a DO statement and its final value (plus the increment for normal termination of the DO loop) is always stored there.

Examples: DO 42 K = 1,L
DO 3 JT = MIN, 55, NSTEP

9. CONTINUE

This is a dummy statement that generates no object code except when it is the last statement in a DO loop. Its statement number is always used if it has one. It must be used as the last statement in a DO loop whenever the last statement would have been an arithmetic IF or GOTO type of statement that transfers control. Whenever a CONTINUE statement is the last statement in a DO loop, its statement number is the location of the machine instructions that increment the DO variable and test it for its maximum value.

10. STOP or STOP w where w is an octal constant that is ignored.

This statement causes the running program to be terminated and should only be used to indicate that the program has run to completion. This statement causes BRLESC FORTRAN to empty the tape output buffers, rewind all tapes used by the program that have not been rewound, check for overflows and halt at N40. On BRLESC, the program may also be terminated by reading a card or tape line that has the first ten characters of either "ENDbTAPEbb" or "bbbbbbPROB" where b represents a blank.

Examples: STOP
STOP 77

11. PAUSE or PAUSE w where w is an octal constant.

This statement causes the program to halt and display the octal constant. (BRLESC displays it in the α address of the halt order.) If the computer is re-started manually by pressing the proper button (initiate on BRLESC), the program will continue with the next statement.

This statement should not be used without a very good reason for using it.

Examples: PAUSE
PAUSE 421

12. CALL a(b,c,d,.....).

This statement causes the subroutine named "a" to be entered and executed with b,c,d,... as the arguments, parameters, and store addresses. (Arithmetic expressions are allowed.) For BRLESC FORTRAN, "a" could be the name of a function (a subroutine with one result) and the subroutine being called must be one that is a standard one on the compiler tape or one whose code is included in the program as a SUBROUTINE (or FUNCTION).

The arguments used in a CALL statement must agree in type with the type of the dummy variables that were used when the sub-

routine was defined. If there are no arguments, they may be omitted.

CALL EXIT or CALL DUMP statements on BRLESC are the same as a STOP statement and CALL PDUMP is ignored.

Alphanumeric arguments of six or less characters are allowed in BRLESC FORTRAN.

Examples: CALL SUB3(X,Y,R)
CALL TOTAL

13. IF (SENSE SWITCH i) s1,s2.

This statement transfers control to statement s1 or s2 if sense switch i ($1 \leq i \leq 6$) is down or up respectively. (i must be a constant.) On BRLESC, the manual read switches 15-20 are used as sense switches 1-6 respectively. However, these switches may be "preset" by a program control card to be either "down" or "up" regardless of their actual position. (See SETSSW in Section XVII.)

Example: IF (SENSE SWITCH 3)14,92

FORTRAN IV does not usually allow this statement. Instead a subroutine SSWTCH is predefined. The general form of its use is

CALL SSWTCH (i,j)

where i is the number of the sense switch to be tested and j is set to 1 if it is down and j is set to 2 if it is up.

14. SENSE LIGHT i where i is 0,1,2,3, or 4.

If i is 0, then all sense lights are turned off. If $1 \leq i \leq 4$ (actually 6 on BRLESC), sense light i only will be turned on. The rightmost four (actually six) bits of cell 062 on BRLESC are used as sense lights. Initially on BRLESC all of them are off.

Example: SENSE LIGHT 2

FORTRAN IV does not usually allow this statement. Instead a subroutine SLITE is predefined. The general form of its use is

CALL SLITE(i)

where i is the number of the sense light to be turned on. If i = 0, all sense lights are turned off.

15. IF(SENSE LIGHT i)s1,s2

If sense light i is on, it is turned off and statement s1 is done next;otherwise statement s2 is done next.

Example: IF (SENSE LIGHT 2)67,39

FORTRAN IV does not usually allow this statement. Instead a subroutine SLITET is predefined. The general form of its use is

CALL SLITET(i,j)

where i is the number of the sense light to be tested and turned off. If the light was on, j will be set to 1 and if the light was off, j will be set to 2.

16. IF ACCUMULATOR OVERFLOW s1,s2

This statement checks for floating point exponent overflow on BRLESC and does statement s1 next if it has occurred. Otherwise statement s2 is done next. (The very last operation may not be included in the check on BRLESC and this test turns the indicators off if they were on before.)

FORTRAN IV does not usually allow this statement or the IF QUOTIENT OVERFLOW statement. Instead a subroutine OVERFL is predefined. The general form of its use is

CALL OVERFL(j)

where j is set to 1 if the overflow condition was on and j is set to 2 if it was off. The overflow condition is also turned off if it was on.

17. IF QUOTIENT OVERFLOW s1,s2

This does exactly the same as the IF ACCUMULATOR OVERFLOW statement explained above.

18. IF DIVIDE CHECK s1,s2

On BRLESC FORTRAN, this statement checks for floating point division by zero (or unnormalized divisor) or fixed point division overflow. If either has occurred in the program, statement s1 is done next; otherwise s2 is done next. (The very last operation in the previous statement may not be included in this test on BRLESC and this test turns the indicators off if they were on before.)

FORTRAN IV does not usually allow this statement. Instead a subroutine DVCHK is predefined. The general form of its use is

CALL DVCHK(j)

where j is set to 1 on BRLESC if either the floating or fixed point divide overflow condition is on and j is set to 2 if both are off. Both conditions are turned off if they were on.

IX. THE FORMAT STATEMENT

FORMAT (Special Specifications)

This statement is not executed but is used to specify the field lengths, spacing and the form of the data for either the reading of input data or the printing (or punching) of output data. It is always used in conjunction with one of the input-output statements and does nothing by itself.

Let n = number of times to repeat this field. (n is optional, used as 1 if not specified.)

w = the width of the field (the number of columns or characters).

d = the number of decimal places to the right of the decimal point. (d is used modulo 10 on 7090/7094 but not on BRLESC.)

Then the types of fields that may be specified are:

nIw	for integer numbers.
nEw.d	for floating point numbers with exponents.
nFw.d	for floating point numbers without exponents.
wX	for spacing or blank columns.
nAw	for alphanumeric fields.
wH	for alphanumeric (Hollerith) fields where the characters are read into or printed from the w characters following the H in the FORMAT statement itself.
nOw	for Octal numbers.
nLw	for FORTRAN IV logical variables.
nDw	for double precision numbers with exponents.
nGw.d	for generalized floating point numbers.
nRw	for right adjusted alphanumeric fields.

Consecutive field specifications are separated by commas, thus "FORMAT (I6,3E14.6,F10.7)" is an example of a FORMAT statement. Each complete FORMAT statement specifies the maximum length of the record (card or printer line) that will be read, printed or punched when that FORMAT is used.

Two sets of parentheses are allowed in 7090/7094 FORTRAN and four sets are allowed in BRLESC FORTRAN so that groups of specifications may be repeated within a FORMAT statement. A left parenthesis may be preceded by an integer n to indicate the number of times to repeat the specifications enclosed in parentheses. Thus FORMAT (E12.5,3 (I6,F9.3)) would be a format where the I6,F9.3 portion would be repeated three times.

If the input-output statement list contains more items than specified by the FORMAT being used, then a new card or line is begun and the FORMAT is repeated from the left parenthesis that is associated with the next to last right parenthesis. (If there is only one pair of parentheses, then the FORMAT is repeated from the beginning.) If this parenthesis is preceded by a repeat number,

it will be used on most computers including BRLESC. If the FORMAT specifies more fields than required for an input-output list, the rest of the FORMAT is ignored except an H field that follows the last number will be used.

A slash "/" may be used in a FORMAT statement to indicate that a new card or line should be started. Thus FORMAT (I10/E15.6) used for punching cards would cause a ten column integer to be on one card and a fifteen column floating point number to be on the next card. If a slash is used where a new line starts anyway, it is ignored except N+1 consecutive slashes will always cause N blank lines or cards (or skip N cards for input). On some computers but not BRLESC, N slashes at the beginning or end of a FORMAT causes (or ignores) N blank lines.

Scale factors may be used with F type specifications (and in a limited way with E type specifications). An integer, s, specifies the power of ten (scale factor) to multiply the internal number by to obtain the external number, i.e., input numbers get divided by 10^s (not on BRLESC) and output numbers get multiplied by 10^s. The integer s is written in front of the nFw.d specifications and the letter P is used to separate s and n, e.g., -2P4F10.5 or -2PF15.5 specify a scale factor 10^{-2}. On BRLESC FORTRAN, either a + or a - sign in front of s is used as a minus sign. Therefore never write + signs in front of s. Once s has been specified, the scale factor remains in effect for the rest of that FORMAT statement (including repetitions) and will be used on subsequent E and F type fields. A 0P specification may be used to reset it to 0. For input, a punched decimal point overides both the scale factor and the d specified. For E fields, only a positive scale factor may be used and it does not change the value of the number; it only indicates that s digits should be printed in front of the decimal point. (It has no meaning for input E fields.) Thus the number 2 would normally print 0.20E 01 for s = 0, but for s = 1, it would print 2.00E 00 and s = 2 would print 20.00E-01.

I Fields

Input: Most FORTRAN compilers assume the integer to be punched at the right end of the field without a decimal point; however, BRLESC FORTRAN will accept it any place within the field and it may have a decimal point. Any digits following a point are ignored.

Output: The integer will be punched at the right end of the field with a floating sign. (All output has a floating sign which means that the sign is in the column preceding the leftmost digit that is printed. Leading zeros are not printed on I or F fields.)

E Fields

Input: The number may or may not have an exponent. An E or a sign, but not a blank, may be used to indicate the starting of the exponent. The exponent may be less than four columns. If a decimal point is punched, it is used and overides the s and d specification. If no decimal point is punched, then it is assumed to be after d digits (columns) left from the start of the exponent. Most FORTRAN compilers require that the number be punched at the right end of the field, but BRLESC FORTRAN allows it anywhere within the field. Blank columns are used as zeros (except after the exponent on BRLESC).

Output: The floating point number will be printed with a four column exponent that includes an E, a sign, and two digits for the value of the exponent. A decimal point is printed d digits from the right end of the coefficient and if $s = 0$, a zero is printed in front of the decimal point. If $s \geq 1$, then s digits of the

coefficient are printed to the left of the point. The sign immediately precedes the first digit printed. The entire number is printed at the right end of the field of w columns.

F Fields

Input: The same as E fields, see above. (This may not be strictly true for other computers but will generally give the desired result except possibly for the use of a scale factor.)

Output: The floating point number will be printed without an exponent and the decimal point will be printed d digits from the right end of the field. The actual number printed is 10^s times the number that is in the computer. If the number is too large for the columns specified, BRLESC will print the number with an exponent or as much of the right portion of such a number as is permitted by the field width.

H Fields

Input: The alphanumeric information is stored in the FORMAT statement itself immediately following the H. No transformation of characters is done ; the sign option setting for numeric input on BRLESC has no effect on H fields.

Output: The w alphanumeric characters that immediately follow the H are printed. Blanks are not ignored and there is no transformation of any of these characters. Thus on BRLESC, "(+ - " characters may not be the ones intended if the deck was punched using standard FORTRAN characters at some other installation. (The

"CHANGE + AND -" control card does not change the + and - signs in H fields.) For tape output, if an H field occurs at the beginning of a line, the first character is used for vertical high speed printer format control instead of actually getting printed.

A Fields

Input: If $w \leq 6$, this causes w alphanumeric characters to be stored in the variable name that is on the input list. To be compatible with 7090/7094, BRLESC FORTRAN stores a maximum of six characters per word at the right end of the word. If $w < 6$, the characters will be at the left of the 36 bits with blanks to fill out the word. If $w > 6$, then $w - 6$ columns will be ignored before storing the rightmost six characters of the field. As with H fields, no transformation of characters is done. This can be used to read FORMAT specifications at run time.

Output: This causes w alphanumeric characters to be printed from the contents of the variable name that is on the output list. The rules listed above for A input are followed so that whatever is read will be printed exactly the same. When $w > 6$, $w - 6$ blank columns will be printed to the left of the six characters that are printed.

X Fields

Input: This causes w columns to be skipped whether they are blank or not.

Output: Causes w blank columns to be printed.

O Fields

Input: This allows octal numbers to be read and stored at the right end of BRLESC words in the same manner as integers. (There is no left normalization.) On BRLESC, if $w > 12$, the leading columns will be used and may cause more than 36 bits to be stored if they are not blank.

Output: This allows integers (octal or decimal) to be printed in octal form at the right end of the field with leading zeros suppressed. If w > 12, w-12 blank columns are printed to the left of the 12 octal digits.

L Fields

Input: If the first non-blank character is a T (or the digit 1 on BRLESC), the logical value .TRUE. is stored; otherwise .FALSE. is stored.

Output: A T is printed in the rightmost column of the field if the value of the logical variable is .TRUE.; otherwise an F is printed in the rightmost column of the field.

D Fields

Input & Output: This is allowed for those computers that use double precision variables. On BRLESC, it is used exactly the same as an E field.

G Fields:

Input & Output: BRLESC uses G fields exactly the same as F fields.

R Fields

Input & Output: Only a few computers and BRLESC allow R fields. They are exactly like A fields except when w < 6, the characters are stored into (or printed from) the right end of the computer word.

FORMAT statements may be placed anywhere within a program (or subprogram) except as the first statement within a DO loop. (This restriction does not apply to BRLESC but should be followed. On BRLESC, FORMAT statements are done as NOP instructions so it is best not to place them where they will be done often.) FORMAT statements are kept as alphanumeric information and decoded at run time, thus it is permissible to use A fields to read FORMAT statements (without the word FORMAT) at run time. The variable names of such statements must be listed in a DIMENSION statement for most computers but is not required for BRLESC.

If the list in an output list is exhausted and the next item in a FORMAT statement is an H field, the H field is printed. (If the end of FORMAT and list occur at the same time and an H field follows the last left parenthesis, it will not be printed.) Note that a FORMAT may contain nothing but one or more H fields.

Blank characters in a FORMAT statement are ignored except within H fields. The w count for an H field must include the blanks within the H field.

The comma separating field specifications may be omitted when it follows an H or X field specification or would precede or follow a parenthesis or slash. (This rule may not hold for all computers but is true for BRLESC.)

Examples: FORMAT(3I5,(E15.8))
FORMAT(2HX=,F10.4,4(1PE12.5))
FORMAT(6F10.4/4I10//)

X. DESCRIPTION OF INPUT-OUTPUT LISTS

The names of the variables to be transmitted between the computer and the input-output devices are specified on a list in the proper type of input-output statement and the sequence of the names on the list determines the sequence of transmission. Simple variable names, subscripted array names where the subscript control is either specified in other statements or within the input-output list, and array names without subscripts are allowed. Array names without subscripts cause the entire array to be transmitted and the elements must (for input) or will (for output) be arranged in the same sequence that they have in the computer memory. (BRLESC and most computers vary the subscripts from left to right, thus two dimensional arrays are stored by columns; i.e. A(1,1), A(2,1), A(3,1) etc. is the sequence of elements of the array A.) Commas are used to separate the names on an input-output list.

Indexing information specified within the list is written after the names of variables to which it applies and the names and the indexing information are all enclosed in parentheses. For example A, (B(I), I = 1,10) would cause the transmission of A, B(1), B(2), ... B(10). Note that the indexing information is written the same as in a DO statement with the increment taken as one if it is not written. It is permissible to nest these parentheses, e.g., ((A(I,J),I = 1,5), J = 1,5). Note that commas are used to separate items on the list and must be used after a right parenthesis except for the last one. The indexing within each set of parentheses is done to completion before going on to the next indexing specification. On BRLESC, there is a restriction that when indexes are controlled within an I/O list, they cannot be used in any subscript arithmetic expression that requires more than the addition or subtraction of a constant.

All of the input-output statements that transfer alphanumeric (not binary) data make use of a FORMAT statement to specify the field types and lengths. The type (integer or floating point) of a name specified on an input-output list must correspond to the type of field specified in the FORMAT statement that is being used. All integer variables must use I fields and all floating point variables must use E or F fields. (BRLESC does allow integers to be printed as integers in E or F fields.) The FORMAT controls the maximum length of each line. A line is shorter than specified in a FORMAT only when the end of the list is reached before the end of the FORMAT. Whenever the end of the FORMAT is reached before the end of the list, the FORMAT is repeated from the left parenthesis that is associated with the next to last right parenthesis and a new line (or card) is started. (If there is only one pair of parentheses, then the FORMAT is repeated from the beginning.) (See Section IX for more information about FORMAT statements.)

Constants and arithmetic expressions are not permitted on input-output lists, except indexing information may contain constants and subscripts may be constant or arithmetic expressions.

It is permissible to read an integer variable and use it as a sub-

script within the same input list if its name is separated from the place it is used by at least two left parentheses. (This is counting the one used to indicate a subscripted variable. Extra parentheses may be used just to meet this requirement.) Thus J,(B(J)) is an example where the value of the variable J just read will be used as the subscript for B(J). (For BRLESC, the extra parentheses are not required if two or more variables or any indexing information separates the integer from where it is used.)

Examples: A, B, I
N, M, (BA(N)),P
((A(I,J), J = 1,10), I = 1,10), (R(K), K = 2,20,2)

XI. INPUT-OUTPUT STATEMENTS

The following group of statements may be used in FORTRAN to control the flow of information between the computer and input-output devices or secondary storage. Card reading or punching, magnetic tapes and, on some computers but not on BRLESC, drums may be used to read or record data. Most of the statements also use a FORMAT statement to control the conversion of data between computer form and printer or card form. However, the READ TAPE or READ(t) and WRITE TAPE or WRITE(t) (and the corresponding DRUM statements on computers that allow them) cause the transfer of data without any conversion. This computer form of data will be referred to as binary information and actually is binary numbers for a binary computer such as BRLESC. The other statements cause the reading or printing of data in alphanumerical form. There are three statements, END FILE, REWIND and BACKSPACE that do not transfer data but can be used to manipulate the magnetic tapes.

In all of the input-output statements described below:

f	is a FORMAT statement number or name.
"list"	is any allowable input-output list (See Section X).
t	is a magnetic tape number or integer variable. (See BRLESC restrictions on t at end of this section.)

READ f, list

This statement causes decimal and alphanumeric data to be read from cards (or tape 6 on BRLESC if the cards have been put on tape off-line and console switch 36 is up.) (BRLESC may use all 80 columns for either input or output cards.) If the list is omitted on this statement, one card will be read and ignored on BRLESC.

PUNCH f, list

This statement causes decimal and alphanumeric data to be punched on cards (or actual tape 8 if console switch 35 is up. The tape output will be "formatted" for the high speed printer by adding a 1 character at the beginning of each "card" and an end-of-line character at the end of each "card". The block length will be at least 1830 characters.) All 80 columns of a card may be used on BRLESC and for tape 8 output, the "card" may be up to 160 columns long.

PRINT f, list

For most computers, this statement means to print the data on an on-line printer. Since BRLESC does not have an on-line printer, the data is put on actual tape 8 for off-line printing. The maximum line length for most computers is 132 characters and it is best to use 132 as maximum although BRL does currently have one printer that has 160 columns.

The following description generally applies only for BRLESC. If the first character of a line comes from an H field, it will be used for vertical format control (after a transformation) and not printed. If the first character does not come from an H field, an extra "1" character (single space) is inserted at the beginning of the line.

In either case, the zone bits will be set to 01 so that it is possible to print PRINT output separately from PUNCH output when both are on tape 8. The end-of-line character is automatically inserted at the end of each line. The tape writing is parity checked and there is checking for end of reel. The tape block length is at least 1830 characters and this allows about 7.5 million characters on 90,000 lines of 80 characters each on a reel of BRLESC tape.

For BRLESC, a control card may be used to change all PRINT statements to PUNCH statements.

READ INPUT TAPE t, f, list
READ(t,f) list (FORTRAN IV form.)

These statements cause decimal and alphanumeric input data to be read from tape t. Each block of BRLESC tape may be as long as 2000 characters and each line may be as long as 160 characters. If the tape was previous FORTRAN output that has a vertical control character at the beginning of each line, provision should be made in the FORMAT for skipping that character. However on BRLESC, the vertical control character is ignored unless the FORMAT has an H field at the beginning of the line. (If the tape was previous FORAST output, the vertical control character is automatically ignored.)

The tape reading is parity checked and there is checking for end of reel.

If the "list" is omitted with this statement, it will cause one line to be read and ignored on BRLESC.

Just INPUT may be used instead of READ INPUT TAPE in this statement on BRLESC.

WRITE OUTPUT TAPE t,f, list
WRITE (t,f) list (FORTRAN IV form.)

These statements cause decimal and alphanumeric output data to be recorded on tape t. Each line of data may not exceed a total of 132 characters (160 on BRLESC). The first character will be used as a

vertical control character for the high-speed printer and is determined in the same manner as for the PRINT statement described above except the zone bits will be 00.

The tape writing is parity checked on BRLESC and there is checking for the end of a reel. The number of lines per reel on BRLESC will vary from about 60,000 to 200,000 as the length of each line varies from 160 characters to 1 character.

Just OUTPUT may be used instead of WRITE OUTPUT TAPE in this statement on BRLESC.

READ TAPE t, list
READ(t) list (FORTRAN IV form.)

These statements cause binary information to be read from tape unit t. It should be used only for reading data that was previously put on tape by the use of the WRITE TAPE statement described below. This statement will not read more data than was specified on the list of the statement that wrote the data. (Such a group of data is defined to be a "logical record".) If less than the entire logical record is read, the tape will move to the end of the record. (If the list is omitted entirely, the tape still moves to the next logical record.) If an attempt is made to read more data than is on one logical record, the unused portion of the list will be ignored.

On BRLESC, binary logical records are subdivided into tape blocks of 128 words each. Within each logical record, the first word of each block contains in the α address the number of words in the last block (not counting this word) and in the γ address, the total number of blocks in the logical record.

WRITE TAPE t, list
WRITE(t) list (FORTRAN IV form.)

These statements cause all of the data specified on the list to be written as binary information in one logical record (see READ TAPE) on tape unit t. It is useful for temporarily recording data on tape that may be read back into the computer by using a READ TAPE statement at a later

time. See the explanation of the READ TAPE statement for a description of the way the information is "blocked" on the tape.

END FILE t

This statement causes a file mark to be written on tape t.

BACKSPACE t

This statement causes tape t to be moved backward one "logical record". This is all of the data written by the WRITE TAPE statement that wrote the record for a binary tape, or is one line (or "card") if it is an alphanumeric tape.

REWIND t

This statement causes tape t to be rewound without being interlocked.

READ DRUM i, j, list

This statement is not allowed on BRLESC and causes an error print. For the 709/7090, it means to read data from drum i beginning at the jth word. (Variable subscripts are not allowed on the list.)

WRITE DRUM i, j, list

This statement is not allowed on BRLESC and causes an error print. For the 709/7090, it means to write data on drum i beginning at the jth word. (Variable subscripts are not allowed on the list.)

Additional Notes on Input-Output Statements

The f (FORMAT number or name) may be omitted in READ, PUNCH or PRINT statements on BRLESC and this will cause a FORMAT (1P6E12.5) to be used automatically.

The statement numbers 1 and 2 may be used to automatically specify FORMAT (5F14.5) and FORMAT (1P5E14.5) respectively without including them as part of the program. If 1 or 2 or both are used to refer to these FORMATS, then that statement number must not be used in the program for any other purpose. If either one is used as a statement number in a program, then the corresponding automatic FORMAT cannot be used.

The omission of a "list" on any of the input statements will cause one record (card, line, or logical binary tape record) to be read and ignored on BRLESC. Some computers may allow the FORMAT specified to be used to skip more than one record. Note that a FORMAT should be specified when the list is omitted although it is not necessary to do so on BRLESC.

The number of print positions on one of BRL's printers is 160. However it is best to restrict the line length to 132 characters or less so that other printers may also be used. BRLESC FORTRAN allows at most a total of 170 characters for a line including the vertical control and end-of-line characters

ADDITIONAL NOTES ON THE USAGE OF MAGNETIC TAPE ON BRLESC:

All of the tape reading and writing is parity checked. Rereading erroneously ten consecutive times or rewriting wrong twice after each of five consecutive "GAP instructions" causes an error print and BRLESC stops running the programs.

There is checking for end-of-reel. At the end of a reel, BRLESC halts at 080 and is ready to accept a new reel when re-started. A single reel of BRLESC tape will hold about 90,000 lines.

The only restriction on switching between reading and writing of tapes is that a REWIND or BACKSPACE statement must be done before reading a tape that was just written. Whenever a tape on BRLESC is switched from writing to reading, a file mark and an extra one word block that says "END TAPE" is automatically written on the tape before the final file mark is written and then switching is done. (This extra block is ignored by a BACKSPACE statement.)

All FORTRAN alphanumeric input and output tapes are "buffered" and may contain up to 2000 characters per block. To accomplish this buffering, each tape unit used requires the use of an extra 201 words of BRLESC memory. This space is assigned as it is needed while the program is being executed and will not conflict with any other memory

assignment made in a normal FORTRAN program. Buffers are assigned backward from the subroutines as long as space is available there. Otherwise they are assigned backward from the end of the memory. (For "CHAIN jobs", they are assigned so as to not conflict with any link of the CHAIN.)

The tape unit number t may be either a decimal integer constant or variable. If t is a variable, the integer value it has at the time the tape statement is executed is used as t. The following table shows the correspondence between the value of t and the tape switch on BRLESC. The actual tape handler used depends on the switch setting.

t	Switch
1 or 11	1
2 or 12	2
3 or 13	3
4 or 14	4
5 or 15	5
6	9
7	10
8	11
9	12
10	7 (temporary or output only)

Note that $1 \leq t \leq 15$ and that t is used modulo ten for $11 \leq t \leq 15$. It is illegal to use both 1 and 11, or 2 and 12,...,or 5 and 15 within the same program. (If $t > 15$ is used, it will be used modulo 16 with 0 using Switch 13. PRINT (and PUNCH) tape output uses Switch 8, the compiler itself uses Switch 14 or 15 for its own program and uses Switch 7 for temporary storage while compiling, and card input put on tape off-line uses Switch 6. (Usage of switches 6 and 7 are identical to FORAST.) When leaving problems to be run on BRLESC, the switch number rather than the t number should be used in the instructions to the computer operator.

All printer output is formatted for variable length lines for the off-line high speed printer. PUNCH tape output automatically has a single space character "1" inserted at the beginning of each line and an end-of-line character at the end of each line. The same is true of PRINT and WRITE OUTPUT TAPE output if the first field of the line is not an H type field. If the first field is an H field, then the first character of the field is used for vertical format control after undergoing the following transformation:

H Field	Tape
+ or -	blank (no space)
0 (zero)	special blank line with 1 on next line. (is double space)
2	2 skip to even numbered line. (possible double space)
1 or 8	8 (start new page)
others	1 (single space)

A blank control character should normally be used to obtain single spaced lines. The zone bits of the format character will be 00 except PRINT output will have 01. When reading previous FORTRAN alphanumeric output, a 1 vertical control character is transformed back to blank and the special blank line is ignored but causes the 1 control character on the following line to be transformed back to zero. The special blank line is also ignored by the BACKSPACE statement.

FORTRAN programs are supposed to contain an END FILE statement and a REWIND statement for each output tape used in the program and a REWIND statement for each input tape. If this is not done within the program, BRLESC will rewind all tapes that were used and not rewound by the program.

XII. DATA STATEMENT

The DATA statement of FORTRAN IV allows initial values to be stored for variables without writing an executable formula. The DATA statement allows a list of variable names to be followed by a list of the constants that should be initially stored as the values of the variables. A slash is used to separate a list of variables and a list of constants and commas are used to separate items within both lists.

The general form of the DATA statement is

DATA v1,v2,v3,.../c1,c2,c3,.../,v4,v5,.../c4,c5,.../,.....

where v1,v2,... represents names of variables and c1,c2,... represents constants. The variable list may contain DO-implying parentheses with variable subscripts that take on specified integer constants. All other subscripts must be constant, i.e.,the integer value of all subscripts must be completely defined within the DATA statement. The name of an array may be used without subscripts to specify a list of the entire array.

The constant list may contain any standard FORTRAN constant and may also contain octal constants by preceding the octal digits with the letter O. T and TRUE are also allowed for .TRUE. and F and FALSE are also allowed for .FALSE.. Any constant may be repeated k times by preceding it with "k*" where k is the integer number of times that the constant should be repeated.

Most computers will not allow the DATA statement to initialize a variable that is in blank COMMON; however this is allowed on BRLESC. Also most other computers allow variables in labeled COMMON to appear in DATA statements only within a special BLOCK DATA subprogram.

Some examples of DATA statements are:

```
DATA A,B/5.3,.6E-3/
DATA I,LOGIC,OCT/14.,FALSE.,07777/,ALPH/4HDONE/
DATA(C(I),I=1,10)/5*1.0,3*2.0,2**3.0/
```

Note the absence of a comma after DATA but the presence of a comma before the beginning of any other list of variable names in the same statement.

There must be a one-to-one correspondence between the number of variables that are to be given initial values and the number of constants within any one DATA statement. BRLESC gives an error print when there is not a one-to-one correspondence.

The form of the constant determines the type of constant stored rather than the name of the variable. There is no check between the type of constant and the type of variable.

To allow some compatibility with CDC FORTRAN, BRLESC also allows the CDC form of the DATA statement which has the general form of:

```
DATA(v1 = c1),(v2 = c2), ....
```

where v1 is one variable name or one array name or one subscripted name, which may have DO-implying subscript information, and c1 is one constant or enough constants, separated by commas, to satisfy the requirements for v1. Repetition of one or more constants k times is allowed by "k(c1,c2,...)".

Some examples of CDC DATA statements are:

```
DATA(F=7.2),(X=.003)
DATA((B(J),J=1,5)=1.0,2(5.3,8.1)),(LA=.TRUE.)
```

XIII. SUBPROGRAM STATEMENTS

FORTRAN allows sections of a FORTRAN program to be designated as subroutines that may be used at many different places in the main program. The SUBROUTINE, FUNCTION, RETURN and END statements allow the programmer to define and name portions of his program as subprograms and they provide information that allows the compiler to provide for the substitution of variables at run time and standard entry and exit methods used for subroutines.

Any subprogram may use any of the FORTRAN statements within itself except SUBROUTINE and FUNCTION statements. Any subprogram may use any other subprogram or subroutine of any type, including arithmetic statement functions (See Section XIV) that are defined at the beginning of the subprogram. Recursive subprograms (subprograms that use themselves) are not allowed.

SUBROUTINE AND ADJUSTABLE DIMENSIONS

SUBROUTINE a(b,c,d,e,...)

This statement marks the beginning of a subprogram that we shall call a SUBROUTINE. The name of the SUBROUTINE is a and b,c,d,e,... are the names of nonsubscripted dummy variables that will be replaced at run time by the actual variables that are listed in the CALL statement that causes this subroutine to be performed. The subroutine consists of the FORTRAN statements that follow this statement down to an END, FUNCTION or another SUBROUTINE statement.

The name of the SUBROUTINE does not indicate the type of any result and hence any letter may be used as its first character.

Except for the COMMON storage, all variables within a SUBROUTINE (or FUNCTION) are assigned storage that is unique and not used by any other part of the program. Thus the variable X may be used in several SUBROUTINES within a program and each X will be different unless it appears in the same relative position in COMMON statements in each of the SUBROUTINES.

No storage is assigned to the dummy variables; on BRLESC, DM will appear in the dictionary instead of an address. The type of a dummy variable, as indicated by its first letter or within a TYPE statement, must agree with the type of all actual arguments that replace it. If a dummy variable is an array name, then its size must be defined within the subroutine and the size must either be identical with the size of any actual variable used to replace it or else its size must be defined by other dummy variables. Dummy variable array subscripts in DIMENSION statements is a feature of FORTRAN IV and is not allowed in other versions of FORTRAN. This feature is commonly referred to as adjustable dimensions. It has the restriction that the dimensions specified in the calling sequence must actually be the maximum dimensions as declared in the calling program. Thus if the array A is declared to be (10x10), but actually has a (5x5) matrix stored there, then this 5x5 matrix cannot be used as an argument for a subprogram. (Most FORTRAN manuals indicate that it is legal to use less than the maximum dimensions as arguments and indeed there is no error check for this, however a normal FORTRAN subprogram will reference the wrong elements of the array when this is done.) However it is true on BRLESC (and probably any other computer that stores arrays by columns) that the last or rightmost dimension of the array does not have to be the maximum. Thus for the array A(10,10), the dimensions of (10x5) could be used as arguments of a subprogram. In particular, the dimension argument for a one-dimensional array does not have to be the maximum value.

Example: SUBROUTINE POGO(A,XX,LO)

FUNCTION a(b,c,d,...)

This statement is similar to the SUBROUTINE statement but should be used whenever the subroutine has only one result. No dummy variable should be listed for the result as it is intended that the function will be used in an arithmetic expression and the result is simply used in evaluating the rest of the expression.

The name of the function is a and b,c,d,... represent nonsubscripted dummy variables. The name of the function does indicate the type of the result by its first letter and the final character must not be F if there are more than three characters in the name. For FORTRAN II, the first character of the name "a" must be I,J,K,L,M or N if and only if the result is integer. For FORTRAN IV, the type of the result may be declared before the word FUNCTION, e.g., REAL FUNCTION, LOGICAL FUNCTION, etc. The type of other dummy variables can be specified in a TYPE statement within the subprogram.

Within the FUNCTION subprogram, some statement should store a value in a variable that has the same name as the name of the function and this will be used as the result.

There must always be at least one dummy variable for FUNCTION subprograms.

Example:
```
FUNCTION LOW(Q1,T)
LOGICAL FUNCTION FOUND (L,V,N)
```

RETURN

This statement may be used as often as desired within subprograms (SUBROUTINE or FUNCTION) to indicate the point or points at which execution of the subprogram should stop and control should return to the program that is using the subprogram. It should always be used at least once in every subprogram.

END

This statement may be used at the end of any subprogram or at the end of the main program. It is not required on BRLESC. All program decks on BRLESC do require the very last card of the entire program deck to be a card that has an E in column 1 or an * in column 1 with DATA in the statement field.

The sense switch options allowed in END statements on some computers will be ignored on BRLESC.

Most computers other than BRLESC compile each subprogram as though it is a complete FORTRAN program and only provide a binary card deck that must be assembled with other binary decks to actually run the program. Hence they require an END statement at the end of each main program and each subprogram.

For BRLESC, the main program and all the subprograms must be compiled at the same time and thus can be run without the use of any binary decks. BRLESC has a limit of 60 subprograms used in any one program deck.

ENTRY a(b,c,d,e,...)

The purpose of this statement is to allow multiple entry points within subprograms. It is not a standard FORTRAN statement but some form of it is allowed in a number of FORTRAN IV compilers. The following description applies only to BRLESC.

The name of the entry point is a and b,c,d,e,.... are the names of non-subscripted dummy variables. The name of the entry point a is used in a CALL statement for ENTRY statements in SUBROUTINES and is used in arithmetic expressions for ENTRY statements in FUNCTION subprograms.

The dummy variables in an ENTRY statement do not have to be the same as those in the SUBROUTINE or FUNCTION statement for the subprogram in which the ENTRY statement appears. However, a dummy argument may not appear in <u>any</u> statement (including DIMENSION) unless it has previously been declared to be a dummy variable by appearing in a SUBROUTINE, FUNCTION or ENTRY statement. On BRLESC, the ENTRY statement must also physically precede all of the appearances of any of the dummy variables that will actually be used in executable statements for that entry to the subprogram. (This is the only essential difference between 7090/7094 FORTRAN and BRLESC ENTRY statements. 7090/7094 FORTRAN allows dummy variables to be used both before and after the ENTRY statement.)

The name of the result in a FUNCTION subprogram cannot be an ENTRY name. Only the name appearing in the FUNCTION statement is allowed as the name of the result.

ENTRY statements are non-executable and normal control may pass through them without doing the initializing of the arguments for that ENTRY.

BRLESC has a limit of 100 dummy variables and ENTRY names in ENTRY statements within one complete program.

CDC FORTRAN 66 allows ENTRY statements without dummy variables. It uses the original dummy variables automatically with each ENTRY.

Example: ENTRY TRY2(V,R)

BLOCK DATA

Most FORTRAN compilers do not allow the DATA statement to store constants into variables that are in labeled COMMON unless an extra subprogram that begins with a BLOCK DATA statement and contains the necessary declarations and one or more DATA statements is used. This subprogram must not contain any executable statements. It must contain one or more COMMON statements that list all of the names that are in any of the labeled COMMON groups that is to receive constants from a DATA statement. It is not permissible on most computers for any DATA statement to store into a blank COMMON variable; however this is allowed on BRLESC.

The use of BLOCK DATA is not necessary on BRLESC but it should be used to maintain compatibility with other computers. BLOCKD will be used as the name of the BLOCK DATA subprogram in a BRLESC dictionary.

Example:

```
BLOCK DATA
DIMENSION A(6)
LOGICAL LA
COMMON/B1/R,A/B2/V,LA
DATA LA,A/.TRUE.,6*1.0/
END
```

XIV. PREDEFINED FUNCTIONS AND ARITHMETIC STATEMENT FUNCTIONS

FORTRAN subroutines are separated into two classes, (1) functions are those subroutines that have only one result and hence may be used in arithmetical expressions; and (2) SUBROUTINE subprograms (See Section XIII) or other subroutines that may have more than one number as a result and may be used only by CALL statements.

Functions

There are three methods of defining a function. They are

1. Predefined functions that may be used by using the predefined name.
2. Arithmetic statement functions.
3. FUNCTION subprograms. (See Section XIII)

Predefined Functions

Appendix A lists the predefined functions that are allowed on BRLESC and most computers. Both the FORTRAN II and IV names are listed for each function and either name is allowed on BRLESC.

Naming Functions

For FORTRAN II, predefined function (and arithmetic statement function)names must always end with F (a total of seven characters are allowed) and must begin with X only if the result is an integer. Variables must never be given a name that is the same as any of the function or subroutine names either with or without the terminal F. For BRLESC, the terminal F is not necessary when the initial letter of the predefined function name indicates the proper type of result but is necessary in both the definition and use of arithmetic statement functions.

For FORTRAN IV, all function names indicate the type of result in the same manner as other variable names, i.e.,either the initial letter determines the type or the type is declared in a TYPE or FUNCTION statement. For both FORTRAN II and IV, the naming of FUNCTION subprogram functions uses rules that are the same as for naming arrays. An initial letter of I,J,K,L,M or N indicates an integer result and the last character must not be F if there are more than three characters in the name. For FORTRAN IV, a type declaration may precede the word FUNCTION in a FUNCTION statement, e.g.,INTEGER FUNCTION RAY(V).

Use of Functions

Any of the three types of functions may be used in an arithmetic expression by writing its name in front of a pair of parentheses that enclose the list of arguments. The arguments must correspond in type and number to the dummy variables used in defining the function. Successive arguments are separated by commas and they may be arithmetic expressions.

For BRLESC, any function may also be used in a CALL statement by including an extra variable name that specifies where to store the result.

Arithmetic Statement Functions

Arithmetic statement functions are functions that can be and are defined by one arithmetic statement at the beginning of a program (or subprogram). The name of the function followed by the dummy arguments enclosed in parentheses are written to the left of the = symbol. The arithmetic expression that describes the function in terms of the dummy variables is written to right of the = symbol. The dummy variables cannot be subscripted. Any variable used in the expression that is not a dummy variable will be identical to the variable of the same name in the program (or subprogram) in which the statement is contained. (An arithmetic statement function definition normally only applies and can be used only in the program or subprogram in which it is located, however BRLESC allows them to be used anywhere within the complete program.)

An arithmetic statement function may use any of the other types of functions and may also use other previously defined arithmetic statement functions. All arithmetic statement functions must precede the first statement that gets executed in the program or subprogram.

If the arithmetic statement function name does not indicate the proper type of result, then its name must appear in a TYPE statement. When an arithmetic statement function name appears in a TYPE statement on BRLESC, it must also be put in an EXTERNAL statement that appears after the TYPE statement and this pair of statements must appear in every subprogram that uses the arithmetic statement function. Note that on BRLESC, an F as the 4th,5th,6th, or 7th and last character in the name causes a leading X or its absence to indicate integer or real result respectively while the absence of such a final F causes the normal I-N check on the leading character. However the usage of a TYPE statement will override both of these checks.

The dummy variable names used must indicate the same type of arithmetic that is required when the function is actually used. When the initial letter of a dummy variable does not indicate the proper type, it may appear in a TYPE statement before the arithmetic statement function. When this is done, the BRLESC dictionary will not have the variable marked as a dummy variable, but the program will be correct.

Example of defining an arithmetic statement function:

```
FUN(A,B,C) = A**2 - SIN(B*C)+C
```

Example of using this arithmetic statement function:

```
T = Q + FUN(X,S + EXP(V**2),14.)
```

XV. PREDEFINED SUBROUTINES

A subroutine may be predefined and exist on the compiler tape or it may be defined by a SUBROUTINE subprogram. (See Section XIII.) Subroutines may be given any valid name (no restrictions on the first or last letter) and may only be used by a CALL statement.

The following subroutines are predefined in BRLESC FORTRAN:

SETMSI (j)	Set minus sign for input.
SETPSI (j)	Set plus sign for input. (Not necessary, anything not minus is plus.)
SETMSO (j)	Set minus sign for output.
SETPSO (j)	Set plus sign for output.

where j is an integer constant:

0 means blank.
1 means y(12) punch.
2 means x(11) punch.
3 means x or y punch.

SEXAPR(A,B)	Sexadecimal print from A to B.
BINPUT	Goes to binary input routine after saving a return jump instruction in 073.

POWERS(A,B,C)	Computes C = A**B where B may be integer or fl.pt.
SINCOS(A,B,C)	Computes B = SIN(A) and C = COS(A).

Additional predefined subroutines may be added in the future.

XVI. FORTRAN PROGRAM CARDS

BRLESC FORTRAN uses the same card format for punching FORTRAN programs that is used by other computers.

Columns:

1 - 5	Statement number (integer).
6	Continuation Card if not zero or blank.
7 - 72	One FORTRAN statement.
73 - 80	Identification.

The statement number must be a decimal integer. Leading zeros are ignored. Trailing blanks are also ignored on BRLESC and most computers.

Column 1 is also used to indicate special types of cards. The following list shows the special characters that indicate special cards:

C	Comment card. Columns 2-80 may be used for comments.
*	7090/7094 monitor card or BRLESC control card.
B	Boolean statement card. (FORTRAN II)
D	Double Precision statement card. (FORTRAN II)
I	Complex Arithmetic statement card. (Not allowed on BRLESC.)
F	Used to specify names of subroutines used as arguments. (FORTRAN II)
$	BRLESC only, used for BRLESC assembly order cards and some other special BRLESC cards. (% is used until January 1967.)
7-8	End-file signal on 7090/7094, ignored on BRLESC.
E	BRLESC only, is last card of program deck.

Column 6 is used to mark cards that are a continuation of the previous card. It is used as a continuation if Column 6 contains any character other than zero or blank except for an initial identification control card and all comment cards. BRLESC does not limit the number of continuation cards allowed for one statement but some other computers do have some limit.

Columns 7-72 contains information, one or more statements, comments, control information, etc. depending on the type of cards as indicated by Column 1. BRLESC will allow more than one statement per card if the symbol $ (> until January 1967) is used to separate the statements. A special program is available for compacting and repunching a FORTRAN program so that it will have more than one statement per card.

Columns 73-80 are ignored by BRLESC and may contain any desired identification.

Blank columns are ignored except when they are in an H field in a FORMAT statement or alphanumeric constants.

Blank cards will be ignored on BRLESC.

Note that it is permissible to use FORAST coding sheets to write FORTRAN programs. The key punchers must not use Column 6 as part of the statement number and must not allow a statement to go past Column 72. It does not matter whether the statement starts in Column 7 or 11.

F Cards

If the name of a subroutine or function (either predefined or defined by a subprogram) is used as an argument for another subroutine or function, its name, usually without the terminal F, must appear on a card with an F in Column 1. This F card must be in the program (or subprogram) that uses the subroutine as an argument and may be anywhere within that program.

The names of the subroutines are to start in or beyond Column 7 and are separated by commas.

Example: F SIN, EXP, FUN3, ATAN

On BRLESC, the terminal F is to be omitted from those function names that have an initial letter that indicates the proper type of result according to the I-N rules. It must be retained on those names that do not indicate the proper type of result, e.g., LOGF , MAXOF. This same rule for the terminal F applies where the name is used as an argument for a subprogram. When programming a subprogram to accept a function name as an argument, the dummy variable should end with F only if the initial character does not indicate the proper type of result. If a final F is used with at least three other characters, then the result type is integer only if the name begins with X.

For FORTRAN IV, the EXTERNAL statement replaces the F card and serves the same purpose.

XVII. BRLESC CONTROL CARDS AND DICTIONARY PRINTING

The use of certain control cards are allowed to affect the compilation of FORTRAN programs. Most of these apply to BRLESC FORTRAN only, although some are also used on 7090/7094. All of the control cards are marked with an * in Column 1 with the control information starting in or after Column 7.

* The first card of a program that has an * in Column 1 is used as identification and is printed in front of the PUNCH output. Columns 2-80 may be used. (On all other cards with * in Column 1, only Columns 7-72 may be used.) The first thing after the * should be the official problem number followed by a blank column and this should not extend beyond column 20.

* CHANGE + AND -

This card reverses the meaning of + and - signs in the program deck, except in FORMAT statements. BRL signs are used initially. (After January 1967, this card will not be allowed and will cause an error print.)

* SETSSW i $\left\{\begin{matrix}\text{UP}\\\text{DOWN}\end{matrix}\right\}$

This control "statement" allows sense switch i to be "preset" either UP or DOWN. By using this control card, the operator can be relieved of actually setting the sense switches.

* PRTOPU

This control statement causes the compiler to translate all following PRINT statements as though they were PUNCH statements. (Allows card output instead of tape.)

* RTTORC

This control statement causes the compiler to translate all following READ INPUT TAPE, INPUT or READ(t,f) statements as though they were READ statements. (Use card input instead of tape.)

* WTTOPU

This control statement causes the compiler to translate all following WRITE OUTPUT TAPE, OUTPUT or WRITE(t,f) statements as though they were PUNCH statements.

* WTTOPR

This control card causes the compiler to translate all following WRITE OUTPUT TAPE, OUTPUT or WRITE(t,f) statements as though they were PRINT statements.

* LIST

* SYMBOL TABLE

Either of these causes the storage dictionary to be printed. The asterisk in Column 1 is not required on the LIST card.

The dictionary is printed with names of variables arranged in alphabetical order within each subprogram. Function and subroutine names will be preceded by two asterisks. Main program names will be preceded only by two blanks and subprogram names will be preceded by one character and one asterisk or period. The character preceding each subprogram name will be 1,2,...,9,A,B,...T corresponding to the sequence in which the subprograms appeared in the program deck. The name of each subprogram will appear on a separate line before the dictionary for that subprogram. If more than 30 subprograms are used, some dictionaries for two subprograms will be mixed together with both subprogram names preceding that section of the complete dictionary. When this occurs, those names preceded with an asterisk are from the subprogram whose name appears on the left side of the subprogram name card and those names preceded with a period are from the subprogram whose name appears on the right.

Following each name will be the sexadecimal memory address that has been assigned to the name. Following this address, any of the following letters may appear:

A indicates an array name.

I indicates an integer variable.

L indicates a logical variable. (FORTRAN IV)

C indicates the name was in a COMMON statement.

E indicates the name was in an EQUIVALENCE statement.

U indicates the name was used only once.

Statement numbers are printed at the right end of the six character name position and therefore always precede the names of the variables in any subprogram. The compiler usually adds a few names to the dictionary to indicate temporary storage and special subroutines. The name $SUBS. is printed usually at the end of the dictionary to indicate the length of the predef'ned subroutines. The subroutines extend from this address down through 0103L and includes all of the input-output storage and subroutines. The $NOS. name printed usually as the next to last name in the dictionary· indicates the length of the "constant pool". This storage, from 0S0 down to but not including the address printed after $NOS., is used to store the constants and the "array words" required by the program. The $LAST entry printed with the dictionary indicates the largest address used by the program with the possible exception of some tape buffers at the end of the memory.

For array names, the address printed in the dictionary is the initial address of the array.

The names of all the COMMON variables used within a subprogram may not appear in the dictionary for that subprogram. When the common statements of a subprogram are processed, a check is made to determine if the names and required storage are the same as those for the main program. All of the names up to the point of the first disagreement in name or storage are deleted from the subprogram dictionary. If the subprogram common statements are identical to the main common statements the then the words MAIN COMMON are printed preceding the subprogram dictionary. If the first common name of the subprogram disagrees with the first common name of the main program, then the check and deletion explained above is made with the common statements of the previous subprogram.

If the subprogram common statements are identical to the previous subprogram common statements then the name of the previous subprogram followed by COMMON is printed preceding the subprogram dictionary.

Names in COMMON are assigned last, so the last name in the COMMON assignment within the subprogram that has the most COMMON storage will mark the end of all the storage used by the program. The instructions for the program and all the subprograms are stored first, then all the variables not in COMMON are assigned storage immediately after the instructions and this is followed by those variables in COMMON.

* LIST8

* LIST (S.CODE)

Either of these control cards causes the dictionary and the sexadecimal code for the entire program to be printed. Four instructions are printed on a line with the address of the first one printed at the beginning of the line. The * in Column 1 of LIST (S.CODE) may be omitted unless LIST is the name of an array.

* LIST (B.CODE)

This control card causes the entire program and the subroutines it uses to be punched on binary cards with absolute addresses. To use this deck to run the program, it must be preceded by a binary input routine and followed by the standard set of FORTRAN input-output routines and a jump to 073. The * in Column 1 may be omitted unless LIST is the name of an array.

XVIII. BRLESC ASSEMBLY ORDERS

BRLESC FORTRAN allows BRLESC assembly orders to be written on cards that have a $ in column 1 (% is used instead of $ until January 1967). The same general form as used in FORAST is allowed, but not the special processing of addresses, formulas, etc. FORTRAN statements must not be put on the same card with assembly orders, but more than one assembly order may be put in columns 7-72 by separating them with a $ symbol. The general form of each order is like FORAST, i.e., OT(A)B)C$ where "(" after ")" is optional, the last ")" and the last $ are optional and less than three addresses is permitted. Comments may follow $$.

For BRLESC assembly orders:

Col.1	$ (% is used until January 1967)
Cols.2-5	Blank, statement number or symbolic name is allowed.
Col.6	Normal FORTRAN continuation column. ($ in col.1 not required on continue cards.)
Cols.7-72	One or more BRLESC assembly orders.
Cols.73-80	Identification only.

The following symbolic order types are the only ones allowed:

A	B	SET	J-	JC	MI
S	CB	SI	TAPE	NOP	IM
M	CEQ	INC	CARD	RSW	RCL
D	CNB	II	ZERO	MMF	
C	CNEQ	LP	SLJ	LPI	
SQRT	PMA	J	IIJ	MMB	
SH	IT	JS	EA	JNA	
TP	HALT	J+	JA	JNC	

The symbolic parameters of X,F,A or +,V,R, are allowed with the same meaning as used in FORAST. The X and F cannot be written before the order type and a minus sign is ignored except for the J- order type.

A decimal parameter is also allowed. All arithmetic orders, including SHift, are floating point unless X parameter is used.

A GOTO statement with one address is allowed but none of the other general FORAST statements are allowed.

No assembly order may have more than three addresses including SET and INC and they cannot include a GOTO. However any of the orders that set or increase index registers may be written with an = like FORAST allows. Index registers cannot be increased by a negative amount.

A comma is used to indicate indexing in the same manner as FORAST. Constant increments (and decrements on symbolic addresses) are allowed. All index addresses must be absolute, decimal or sexadecimal. Addresses 13-23(0J-01N) and 48-55(030-037) are not presently used for anything in FORTRAN compiled programs. Dummy variables must never be indexed and variables that might be assigned to a "large address" must not be indexed. "Large Addresses" may be used anywhere as long as they are not indexed. (FORTRAN allows large addressing by using indexing automatically on all large addresses.)

FORTRAN array subscripting is not allowed in any BRLESC assembly orders. When an array name is used by itself, it references the "array word", not the first element of the array.

Decimal addresses are allowed and sexadecimal addresses must have a leading zero. Statement numbers used for an address must either be preceded by "S/" or followed by an S. Decimal numbers must be preceded by an * and may be either a FORTRAN integer or floating point constant. Fixed point fractions are not allowed. Sexadecimal constants may be written following a leading "/" and may use M,A and Z as allowed in FORAST.

There is no special processing of any addresses like there is in FORAST for β of SH,JA,JNA,JC,JNC and γ of I/O orders. These addresses should normally be written in sexadecimal.

Names of subroutines may be used as an address only by preceding each one with "F/" or including each one in an EXTERNAL statement or on an "F card".

Decimal increments and decrements are allowed on symbolic addresses and may be written either before or after the index name, e.g., A+2,14 or A,14+2. SELF is also allowed to refer to an order's own address.

Symbolic names may be assigned absolute addresses by using a SYN statement on a $ card. SYN may be followed by any number of pairs of parentheses that enclose one symbolic address and one absolute address separated by "=". An example is:

$ SYN(A=080)BT=942)(OLOO=Z1)$

Note that "("after")" is optional and that $ at the end indicates that the rest of this card will not be used.

A group of sexadecimal or decimal constants may be stored by using a SEXA or DEC statement on a $ card. Any number of constants may be put on one card but no other statements or assembly orders are allowed on the same card. Constants are separated by parentheses with "(" after ")" being optional. For sexadecimal constants, Z indicates a string of zeros, A a string of five sexadecimal zeros and M a string of five sexadecimal L's. Any legal FORTRAN decimal constant may be written on a DEC card.

Examples: $ SEXA(L)08Z)(10KZ82)4A$

$ DEC(14)3.)6.1E-3)$

Some examples of BRLESC FORTRAN assembly orders are:

$ AV(F)*7.1)T1$ SHX(Q,2)0286)0$

$ TAPE(SS2B)360)017 $ SET(15=0)3=1$

$ INC(2=2+1)21=21+4)$ CNEQ(W)/2A)42S$

$ B12(R-1,4))TT$ TP10)I2)/M)SELF+2$

$ JS(A)A+50)(F/SEXAPR)$ GOTO(TEST)$

$ 029(07000)S,24-1)II$ MMF(1,6)300)08000$

XIX. MAXIMUM TIME AND OUTPUT SPECIFICATIONS

BRLESC FORTRAN allows the programmer to control the maximum amount of time a program will be allowed to run and the maximum amount of output it will be allowed to print. If the programmer does not specify these maximums, BRLESC will set them at five minutes and 1200 lines. Whenever either one of the specified maximums is exceeded, BRLESC will stop the execution of the program after the appropriate error print.

Maximum Time

The maximum time specification is of the form

$ MAXT(integer number)MINS.

where the initial $ is in column 1 (% instead of $ is used until January 1967) and the rest of the specification is in columns 7-72. "MINS." may be replaced with "HRS." or "SECS." to specify hours or seconds instead of minutes. Note that fractions of time units are not allowed.

The time begins when this card is encountered by the compiler and hence some compilation time must be included when estimating the maximum time.

If the statement number 98765 has been used, BRLESC jumps to that statement when the maximum time has been exceeded. If 98765 has not been used as a statement number, BRLESC gives the following error print:

EXCEEDED MAXT. I1= s OCTAL AR.REFS.= a b c CLK= cr

where s is the octal contents of index register 1, a,b, and c are the octal contents of indexes 10,11 and 12 respectively which are the last array addresses referenced, and c r is the clock reading at the time of the error print. This clock reading contains six digits, two each for hours, minutes and hundredths of minutes. This same error print is obtained if BRLESC stops for some reason during execution of a FORTRAN program and the clock reading can be compared with the initial time to determine how long the program ran before it stopped.

Examples:

```
$   MAXT(3)MINS.
$   MAXT(90)SECS.
$   MAXT(2)HRS.
```

Maximum Output

The maximum output specification is of the form

```
$   MAXO(integer number)LINES
```

where the initial $ is in column 1 (% instead of $ is used until January 1967) and the rest of the specification is in columns 7-72. Blank lines caused by slashes in formats count as lines; however any lines skipped by using vertical control characters do not count. All tape or card alphanumeric output is included in the counting of lines but binary tape output is not included.

Note that MAXO ends with the letter O, not zero.

When the specified amount of output has been exceeded, BRLESC will not go to statement number 98765. It gives an error print of:

EXCEEDED MAXO AT s OCTAL AR.REFS.= a b c CLK= cr

where s is the octal address of the statement that caused the output maximum to be exceeded, a,b and c are octal addresses of the last array elements referenced and cr is the clock reading at the time of this error print.

Examples:

```
$    MAXO(500)LINES
$    MAXO(20000)LINES
```

It is permissible for MAXO and MAXT specifications to be on the same card with each other or with BRLESC assembly orders.

XX. STATEMENT NUMBER 98765

The statement number 98765 may be used to obtain some extra printing after a program fails to run to completion or exceeds the maximum time. When an unexpected machine halt occurs, the operator manually causes BRLESC to go to statement 98765 if this statement number was used. At 98765, the program should do a limited amount of printing that could be helpful in determining where and why the program stopped and then should do a STOP statement.

Each link of a CHAIN job may have its own 98765 statement.

XXI. CHAIN JOBS

BRLESC FORTRAN allows segmentation of large programs by using CHAIN control cards and a CHAIN subroutine. BRLESC is essentially compatible with 709/7090 FORTRAN in the way this is done. Each "link" of the chain must be preceded by a control card of

* CHAIN(R,T)

where R is an identifying integer number (less than 32768 on 709/7090) and T is a tape unit designation of any two alphanumeric characters (must be B2,B3,A4 or B1 on 709/7090). BRLESC always uses tape 7 for storing links. Each link consists of a complete FORTRAN program with a main program and all of its subprograms.

Any link may during execution begin execution of any other link by executing a CALL CHAIN(R,T) statement where R and T both are used to identify the link that is to be executed next.

Data may be passed from one link to the next one through COMMON storage only. No program should assume any other storage is preserved from one link to the next. There is a chance of incompatibility between BRLESC and other computers if links that have short programs with long COMMON are mixed with links that have long programs with short COMMON. When this incompatibility arises, an error print of CH.COM.BIF occurs.

Sense switches that are preset with BRLESC control cards will remain preset in all following links unless the link contains a new preset card. Other control cards will not affect following links except a LIST card will cause dictionary printing in all following links.

DATA statements may not be used in chain jobs.

When labeled common is used in chain jobs each link must contain all of the labels. The labels must appear in the same order in each link.

XXII. BRLESC COMPILER ERROR PRINTS

The BRLESC FORTRAN compiler checks for a limited number of types of errors in the program it is compiling. It definitely will not find all possible errors, but some errors will cause one of the error prints listed below. The type of error can be recognized either by the number that follows the word ERROR and precedes the comma or by the "error word" that is printed. The form of the error print is

FORTRAN ERROR t,m Error Word Ident. W First 30 cols. of *Ident.Card

where

t	=	type or error
m	=	ten col. field at which error was detected; m=0,1,...,7
Error word	=	ten alphanumeric characters that describe the type of error as listed below.
Ident.	=	cols.73-80 of card at which error was detected.
W	=	rest of the mth field on the card at time of error detection.

TYPE	ERROR WORD	DESCRIPTION
1	ILL.CHAR.	Illegal character on program card.
2	SYM.ST.NO	Symbolic statement number, not all decimal digits.
3	MIXED EXPR	Mixed expression, integer and fl.pt. arithmetic.
4	INT**FLT	Integer raised to fl.pt. power is illegal.
5	IL.RETURN	Illegal RETURN statement, used in main program.
6	NO = IN DO	No equals symbol at proper place in DO statement.
7	SUBPRS.>60	Tried to compile more than 60 subprograms.
8	BIG ADD.ID	Big address is indexed. Program is too big.
9	NO, CP.GOTO	No comma at proper place in computed GOTO statement.
10	ILL.STAT.	Illegal statement.
11	FLT.INDEX	Subscript involves a fl.pt. number.
12	ILL.DIM.	Number of subscripts is not same as dimensionality of the array.

TYPE	ERROR WORD	DESCRIPTION
13	ILL.COMMA	Comma is used improperly in an arithmetic expression.
14	ASD.ST.NO	Assigned statement number; same statement number used twice.
15	COMPLEX AR	Complex arithmetic cards (I in Column 1) not allowed on BRLESC.
16	EQU. TABLE	EQUIVALENCE table is full.
17	COM. ASGND	COMMON name was previously assigned.
18	ARRAY.REF	Array name used before it was defined.
19	DICT.FULL	Dictionary is full.
20	COL.7 NO.	Statement begins with a decimal digit.
21	SENSE > 6	Sense light or Sense Switch number greater than 6.
22	DO NO END	Statement number used in DO never appeared. (It may have been missed due to another error.)
24	IL.EQUALS	Illegal = symbol or arithmetic was specified on the left of the = symbol.
25	IL. - BOOL	Illegal "not" operation on boolean card.
26	IL. / BOOL	Boolean division is undefined.
28	IL.**BOOL	Boolean exponentiation is undefined.
29	DRUM STAT.	Drum statements not allowed on BRLESC.
30	IL.IO LIST	Illegal input-output list.
31	FAP CODE	An * FAP card is not allowed on BRLESC.
32	BAD TAPE 7	Temporary tape 7 gives persistent parity errors.
33	NO IDENT *	No identification card at beginning of program.
34	N>6 IN NH	Alphanumeric constant of more than six characters.
35	CONST POOL	The constant pool is full.
36	LABEL COMM	More than 32 different COMMON labels.
37	COM. TABLE	The COMMON table is full.
38	STP FULL	More than 800 dummy variable references.
39	DOT FULL	More than 63 nested DO loops.
40	ATP FULL	More than 64 dummy variable references in an arithmetic statement function.

TYPE	ERROR WORD	DESCRIPTION
41	ARG FULL	More than 100 subprogram arguments.
42	FTB FULL	More than 50 subroutine names on F cards.
43	I>32 IN AE	Arithmetic expression has too many operations grouped to the right.
44	ILL.EQUIV	Illegal EQUIVALENCE statement.
45	NON-SEXA.	Illegal character in a sexadecimal word or address.
46	IL.AS.O.T.	Illegal BRLESC assembly order type.
47	IL.AS.ADD	Illegal BRLESC assembly address.
48	NO $ AS.O.	No $ symbol at end of BRLESC assembly order.
49	NON-DEC	Improper character in a decimal number.
50	IL.AS SYN	Illegal SYN statement.
51	DM VAR ID.	Dummy variable was indexed in assembly order.
52	NOT , OR)	Improper punctuation.
53	STPE FULL	More than 600 ENTRY dummy variable references.
54	SEL FULL	More than 100 ENTRY names and dummy variables.
55	DIM. COMMA	Missing comma in DIMENSION.
56	LONG I NO.	Integer constant of more than 17 digits.
57	DUPL. COM.	Duplicated name in COMMON.
58	MAXTO NO I	Number on MAXT or MAXO card is not an integer.
59	EXTRA PUNC	Extra punctuation symbol.
60	BAD L.NAME	Bad logical operation or relation name or illegal period.
61	DATA BAD =	Equal symbol at illegal place in DATA statement.
62	DAT.IL.NO.	Illegal number in a DATA statement.
63	DATA FULL	Too much data in DATA statements.
64	HUNG UP	Computer stopped during compilation.

ERROR. DATA STAT. NOT ONE TO ONE CORRESPONDENCE BETWEEN NAMES AND NUMBERS. (Self explanatory)

FORTRAN ERROR, INDEXED LARGE ADDRESS. 3 PREV. ORDS. + PART ORDER ON NEXT LINE. An assembly order address larger than 16383 was also indexed.

TAPE 7 REACHED END OF TAPE DURING CHAIN COMPILATION PLEASE TRY AGAIN. (Self explanatory.)

ERROR TAPE 7 FORTRAN

If the right end of this error print line says "PARITY ERROR", it was caused by a persistent error on temporary tape 7. If the line ends with anything else, it is a name that cannot be found in the dictionary and this usually indicates a machine error.

It must be remembered that the above mentioned cause is only the probable error. Sometimes some type of undetected error later causes one of the detected error prints at a point where no error exists. It also happens that some errors are not detected until the next card has been read. (W = m = 0 when this happens.)

After each error print, the entire card that the compiler currently has in the memory will also be printed. (If W = m = 0, the error was probably on the previous card, otherwise it probably is the card that contains the error.)

XXIII. BRLESC RUN ERROR PRINTS

Some of the predefined FORTRAN subroutines used on BRLESC detect certain errors in the data they process. When such an error is detected, a RUN ERROR line is printed and the program is not allowed to continue to run. The error print consists of one line of information of the following form:

RUN ERROR "Error word" Date Cols.1-30 of Ident.Card LE No.

where "Error word" is an alphabetic word that identifies the type of error.

Date	is the date.
LE	is the location (in decimal) of the entry to the subroutine that detected the error.
No.	is a number that in some cases was an illegal argument.

Run Error List: (X and Y represent arguments.)

ERROR WORD	SUBROUTINE	REASON	NO.
LOG X NEG	LOGF or ALOG	$X \leq 0$	X
EXP BIG X	EXPF or EXP	$X > 354.89$	$X/\text{Log}_e 2$
ARCSIN 1+	ARCSINF or ARCSIN ARCCOSF or ARCCOS	$X > 1+2^{-49}$	$\|X\|$
SINCOS NS	SINF or COSF or SINCOSF SIN or COS	$\|X\|/2\pi \geq 16^{13}$	$X/2\pi$
POWER 0TO-	POWERS (Exponentiation)	$X = 0$ and $Y \leq 0$	Zero
CVFTOI BIG	XINTF or XFIXF INT or IFIX	$\|X\| \geq 16^{14}$	X

ERROR WORD	EXPLANATION
END TAPE t	Tried to read beyond information written on tape t.
TAPE TKA u	Persistent tape error on trunk A where u is actual tape switch number and "no." is total number of tape errors.
TAPE TKB u	Same as TAPE TKA u except error is on tape trunk B.
BAD FORMAT	Illegal character in a FORMAT statement.
NO(FORMAT	More right parentheses than left parentheses in a FORMAT statement.
LONG LINE	Output line is more than 170 characters.
MACH TAPE-	Machine tape error of setting negative sign bits when there wasn't any parity error.
CH.COM.BIG	Illegal combination of large and small COMMON in CHAIN links.
CHAIN ID.	A CHAIN link called is not on the tape.
CHAIN PAR.	Persistent tape 7 parity error when reading a CHAIN link.
EXCEEDED MAXT	Exceeded maximum execution time or did not run to normal completion. (See Section XIX)
EXCEEDED MAXO	Exceeded maximum amount of output. (See Section XIX)

XXIV. OPERATION OF THE BRLESC FORTRAN COMPILER

The BRLESC FORTRAN compiler exists on magnetic tape in much the same manner as the FORAST compiler and operates in a very similar manner. Many copies of the compiler and the predefined subroutines are on one tape and the tape reading is arranged so that it is checked and automatically corrected by using the next copy on the tape. The tape automatically backs up twenty copies after the last copy on the tape is used. Normally, successive copies are used for compiling successive programs.

Much of the translation is done concurrently with the reading of the program cards (or tape). The partially translated code is put on a temporary tape and the dictionary and constant pool are kept in the memory. After the last card of the program is read (E in Column 1 or * DATA), all unassigned symbols in the dictionary are assigned storage. The memory that will be used by a program is cleared to zeros and then the temporary tape is read, the translation of each instruction is completed and it is stored in the memory for running. Programs are stored from 01040 and may extend to the end of the memory. Next, the subroutines are read from the compiler tape and the ones needed are stored backwards from 0840. (The standard input-output routines occupy 0860-103L.)

The efficiency of the generated code is good except for the referencing of arrays with variable subscripts. Such one dimensional referencing causes one extra order to be done, two dimensional referencing causes two extra orders to be done and three dimensional referencing causes four extra orders to be done. These orders are extra in the sense that they would not be needed in the corresponding FORAST or hand-coded programs. Subscript expressions and other arithmetic expressions are evaluated as they are written except that instructions involving only constants will be done at compile time. The compiler does not presently make use of the "accumulate" option allowed on BRLESC instructions.

XXV. SPEED OF BRLESC FORTRAN COMPILING

The BRLESC FORTRAN compiler is very fast and hence is designed for "load and go" operation. Programmers are encouraged to keep their FORTRAN programs in symbolic form and translate them each time they are run. This wastes very little if any computer time and is most convenient for the programmer.

Most of the translation is done concurrently with reading the program cards at the present maximum speed of 800 cards per minute. The total time required for translating a program consisting of C cards can be approximated by the formula:

$$\text{time in secs.} = 2 + C/13 + C/75$$

The 2 seconds is compiler tape time, the C/13 is card read time and C/75 allows time for reading the temporary tape and completing the translation. If the program to be translated is put on tape off-line, the C/13 term can be reduced by at least one-half. So the translation rate is about 700 statements per minute from cards or about 1500 statements per minute from tape. (The tape rate will vary considerably with the complexity and length of the statements being translated.)

XXVI. RUNNING FORTRAN PROGRAMS ON BRLESC

The following list summarizes the steps for compiling and running FORTRAN programs on BRLESC.

1. Have FORTRAN compiler tape on tape switch 15 (14 if FORAST is on 15.)
2. Have tape switch 7 set to a temporary tape.
3. If have card input, be sure manual read switch 36 is down.
4. If have tape input (program on tape), set manual read switch 36 up and set tape switch 6 to input tape unit.
5. If want all tape output, set manual read switch 35 up and set tape switch 8 to output unit. Also put manual read switch 34 up if this output tape should be rewound at the end of this problem.

6. If programmer specifies any other input or output tapes, mount proper tapes and set proper tape switches. (The programmer may also specify 8 as an output tape without manual read switch 35 being up.)

7. Use "tape start" button to initiate compiling the program.

Halts:

a. N40; End of problem. Initiate only if problems are stacked. Compiler found program error if β address of halt order is LLL.

b. 080; End of output tape reel. Tape unit number is β of halt order. Change reel and initiate.

c. 081; End of input tape reel. Tape unit number is β of halt order. Change reel and initiate.

d. All other halts or cycles; note PO and NI registers and do a jump to 058. It should soon get to N40.

XXVII. MAJOR DIFFERENCES BETWEEN FORAST AND FORTRAN

The following list of some of the basic differences between these two programming languages should be useful to anyone who knows one language and is interested in learning the other one.

1. Statement numbers in FORTRAN must be integer numbers that are used as symbolic names whereas the location field in FORAST may contain any symbolic name and a decimal integer is used as an absolute address.

2. The initial character of a variable name must be alphabetic in FORTRAN and may indicate the type of variable. In FORAST, the initial character may be a decimal digit and has no special significance.

3. The type of names used in FORTRAN arithmetic expressions determines the type of arithmetic performed. In FORAST, the type of arithmetic performed is floating point unless changed by a MODE card or by preceding the formula by "FIX" or "INT".

4. The type of a constant is determined in FORTRAN by the presence or absence of a decimal point. In FORAST, constants assume the same type as the type of the statement they are written in.

5. In FORTRAN arithmetic formulas, automatic conversion from one type of variable to another is provided when the type of variable on the left of the = symbol is different from the type of the variables used on the right of the = symbol. In FORAST, this converion must be accomplished by the explicit use of the appropriate subroutine when it is desired.

6. FORAST allows the use of many = symbols to indicate more than one result address in an arithmetic formula while FORTRAN allows only one variable name for a result address.

7. Constant subscripts are enclosed in parentheses in FORTRAN but not in FORAST.

8. All subscripts have an initial value of one in FORTRAN. In FORAST, the initial value may be specified as zero or any positive integer for each array individually.

9. Variable subscripts are allowed in FORTRAN but not in FORAST. FORAST accomplishes the same thing more efficiently by allowing any address to be indexed by a single index register.

10. Three dimensional arrays are allowed in FORTRAN but not in FORAST.

11. FORTRAN allows a multiply and an add or subtract in a subscript expression while FORAST allows only the addition or subtraction of a constant in an indexing expression.

12. FORTRAN usually allows only one statement per card and FORAST allows more than one. The statement field is columns 7-72 for FORTRAN and columns 11-76 for FORAST.

13. Functions in arithmetic expressions may have more than one argument in FORTRAN but not in FORAST.

14. Implied multiplication is allowed in FORAST but not in FORTRAN (although it does work in some FORTRAN compilers).

15. Absolute addresses are not allowed in FORTRAN but are allowed in FORAST. (They are allowed in BRLESC FORTRAN assembly orders.)

XXVIII. CHECKLIST FOR CONVERTING OTHER COMPUTER FORTRAN PROGRAMS TO BRLESC FORTRAN

1. The first card should be an identification card with an asterisk in Column 1. Columns 2-20 should contain a valid BRLESC problem number.

2. If the signs used in the programs are reversed to BRL usage, insert a "CHANGE + AND -" control card after the identification card.

3. If the signs punched on the input data do not agree with BRL signs, the use of a CALL SETMSI(i) statement is required where i = 1 means y punch is minus, i = 2 means x punch is minus and i = 3 allows either x or y to indicate minus. (The CHANGE + AND - control card does not change input data signs.)

4. DIMENSION, COMMON and EQUIVALENCE statements must be arranged in that order whenever any variable name appears in more than one of these statements. Also a variable cannot be made equivalent to itself, either directly or indirectly.

5. If there isn't an * DATA card between the program and the input data, insert such a card. (A card with an E in column 1 may be used instead of the * DATA card on BRLESC.)

6. If the program uses sense switches, it is best to insert control cards to preset them. (* SETSSW i UP or DOWN)

7. If tapes are used, make sure the tape unit numbers used are compatible with BRLESC. (Those over 9 may need to be changed.)

8. If desired, change tape output to card output or vice versa by inserting control cards. Since BRLESC does not have an on-line printer, PRINT statements cause output on tape 8 unless changed by control cards.

9. Arrays cannot have more than three dimensions and each reference to an array element must use the same number of subscripts as declared in the array definition.

10. A nonsubscripted array name cannot be used to represent only the first element of the array.

11. An array argument for a subprogram must have the same number of dimensions as the corresponding array dummy variable within the subprogram. Such array arguments must not have any subscripts, i.e. the argument must be just the name of the array. For example, it is illegal to try to use a column of a two dimensional array as an argument for a one dimensional array dummy variable.

12. FORMAT statements should not have any + signs on scale factors; BRLESC will use them as minus signs.

13. Alphanumeric constants are restricted to a maximum of six characters except in FORMAT statements. (Also the CHANGE + AND - control card does not interchange sign characters in any alphanumeric constants.)

14. Some computers will skip n lines when n slashes occur at the beginning or end of a FORMAT statement but BRLESC will always skip n-1 lines for n consecutive slashes.

15. When indexing information is specified within an I/O list, BRLESC does not allow these index variables to be involved in arithmetic subscript operations except for the addition or subtraction of a constant. Indexing control within I/O lists does also actually use the variable named rather than just an unnamed index register as happens on some computers.

16. DO indexing variables do actually get used for counting within the DO loop. Some computers may use an unnamed index register instead.

17. The program needs to be modified if it contains any of the following: (1) DRUM statements,(2) I cards (complex arithmetic), (3) assembly instructions for some other computer or, (4) more memory or tape units than available on BRLESC.

18. If the beginning of an assignment statement is the same as the beginning of some other FORTRAN statement, BRLESC may erroneously assume that it is the other statement. One of the error prints will usually occur when this happens. It is best not to use statement names, such as IF, DO, READ, etc., as variable names, especially arrays.

Also TYPER, TYPEI, TYPEL, TYPED and TYPEC must not be used as the first five letters of any statement except a TYPE statement.

19. ENTRY statements may have to be moved closer to the beginning of a subprogram if any of its dummy variables will actually be used in a statement that precedes the ENTRY statement. For CDC FORTRAN programs, a list of dummy variables may need to be added to the ENTRY statement.

20. Non-standard subprogram exits are not allowed. (Indicated by an * instead of a name on an argument list and integer numbers after RETURN in RETURN statements.)

21. Arithmetic statement functions are made available to a whole program rather than just the subprogram in which they appear. This shouldn't cause any difficulty unless the same name has been used for two different arithmetic statement functions.

22. If possible, ask the original programmer if any special characteristics of a particular computer or FORTRAN compiler were assumed when writing the program.

23. If possible, run a test case that has been run on another computer.

In addition to the above general comments, programs that were written for CDC computers can have a number of other incompatibilities. Such programs should be checked for the following items:

1. Assignment statements with more than one = symbol.

2. Mixed type arithmetic expressions.

3. DO loops that will not be done at least once.

4. ENCODE, DECODE, BUFFER IN, BUFFER OUT, IF(EOF,t), IF(IOCHECK,t) and IF(UNIT,t) statements.

5. Executable DATA statements that have statement numbers.

6. Numbers as COMMON labels.

7. Octal constants with a B at the end.

8. ENTRY statements without dummy variables. CDC automatically uses the subprogram dummy variables with each ENTRY statement.

9. Alphanumeric constants with R instead of H preceding the constant.

10. TYPE statements of non-standard form that declare unusual data structures and operations.

XXIX. SUMMARY OF BRLESC FORTRAN IV STATEMENTS

Notations:

s,s1,s2,.....	are statement numbers (look like integer numbers).
i,i1,i2,.....	are integer variable names.
m,m1,m2,.....	are integer variable names or integer constants.
ae	represents an arithmetic expression.
le	represents a logical expression.
b,c,d,e,f	represent any variable names or constants.
t	represents a tape unit number.
f	represents the statement number or variable name of a FORMAT statement.
v,v1,v2	represent variable names.

Specification Statements:

DIMENSION v,v1,v2,....	Defines array names and maximum dimensions of each.
EQUIVALENCE (v,v1,..),(v2,v3,..)	Defines synonymous names.
COMMON v,v1,.../ a /v2,.../ b /v3,..	Defines names common between subprograms. a and b are optional labels.
FREQUENCY s(m,m1,..),s1(m2,..)	Provides optimization information. Ignored by BRLESC.
REAL v,v1,v2,...	Defines real (fl.pt.) variable names.
INTEGER v,v1,v2,...	Defines integer variable names.
LOGICAL v,v1,v2,...	Defines logical variable names.
DOUBLE PRECISION v,v1,...	Same as REAL on BRLESC.
COMPLEX v,v1,v2,....	Not allowed on BRLESC.
TYPE REAL v,v1,v2,...	Same as REAL.
TYPE INTEGER v,v1,v2,...	Same as INTEGER.
TYPE LOGICAL v,v1,v2,...	Same as LOGICAL.
TYPE DOUBLE v,v1,v2,...	Same as DOUBLE PRECISION.
TYPE COMPLEX v,v1,v2,...	Not allowed on BRLESC.
EXTERNAL name1,name2,...	Specifies names of functions or subroutines that are used as arguments for other functions or subroutines.

Assignment Statements:

v = ae	Evaluates expressions ae and stores result in v.
asf(v,v1,...)= ae	Arithmetic statement function where asf represents its name and v,v1,... are the dummy variables.
v = le	Evaluates logical expression le and stores .TRUE. or .FALSE. in v. (v must be a logical variable.) (If the operands for the logical operations in le are arithmetic variables, then the operation is performed on all bits of the word (36 bits on BRLESC). This is a CDC statement.)

Control Statements:

GO TO s	DO statement s next.
ASSIGN s TO i	Put address of s into i.
GO TO i, (s1,s2,...)	Do next the statement whose number was last assigned to i by an ASSIGN statement.
GO TO (s1,s2,...),i	Do statement si next.
DO s i = m1,m2,m3	Repeat statements to and including s with i = m1,m1 + m3, m1 + 2m3,... until i > m2.

DO s i = m1,m2	Same as above with m3 = 1.
IF(ae)s1,s2,s3	Do statement s1 next if ae is negative; s2 next if ae is zero and s3 next if ae is positive.
IF(le)st	where st is any executable statement except IF and DO. Statement st is done only if le has value .TRUE.
IF(ae or le)s1,s2	Statement s1 is done next if the expression is not zero or .TRUE. and statement s2 is done next if the expression is zero or .FALSE. (Is CDC statement.)
CONTINUE	Dummy statement.
STOP or STOP w	End of execution of main program. (w is octal no.)
PAUSE or PAUSE w	Computer halts. (Displays octal no. w.)
CALL name (v,v1,v2,...)	Perform the subroutine specified by "name".
IF(SENSE SWITCH r)s1,s2	Do statement s1 next if switch r is down, do s2 next if it is up.
SENSE LIGHT r	For r = 0 turn all sense lights off. For r = 1,2,3, or 4, turn light r on.

IF(SENSE LIGHT r)s1,s2	Do statement s1 or s2 next if sense light r is on or off respectively. Turn light r off if it was on.
IF ACCUMULATOR OVERFLOW s1,s2 IF QUOTIENT OVERFLOW s1,s2 IF DIVIDE CHECK s1,s2	These are special statements to check certain overflow indicators. Statement s1 or s2 is done next if indicator is on or off respectively.

Subprogram Statements:

SUBROUTINE name (v,v1,v2,...)	Defines the name and beginning of a subroutine. v,v1,v2,... are the dummy variables.
FUNCTION name (v,v1,v2,...)	Defines the name and beginning of a function subprogram.
ENTRY name (v,v1,v2,...)	Define the name and dummy variables of extra entry points for subprograms.
RETURN	Indicates an execution exit of a subprogram.
END	Marks the end of a subprogram.
BLOCK DATA	Special subprogram statement to allow DATA statements to store into labeled COMMON.

Input-Output Statements:

FORMAT (Special Specifications)	Describes the fields for input-output data.
READ f, list	Read cards.
PUNCH f, list	Punch cards.
PRINT f, list	Print data, (on-line on some computers, off line on BRLESC).
READ(t,f) list	Read alphanumeric tape.
READ INPUT TAPE t,f,list	Read alphanumeric tape.
INPUT t,f, list	Read alphanumeric tape.
WRITE(t,f) list	Write alphanumeric tape.
WRITE OUTPUT TAPE t,f,list	Write alphanumeric tape.
OUTPUT t,f,list	Write alphanumeric tape.
READ(t)list	Read binary tape.
READ TAPE t,list	Read binary tape.
WRITE(t)list	Write binary tape.
WRITE TAPE t,list	Write binary tape.
END FILE t	Write end-of-file mark on tape.
BACKSPACE t	Move tape back one record.
REWIND t	Rewind tape.
READ DRUM m,ml,list	Read drum. (Illegal on BRLESC.)
WRITE DRUM m,ml,list	Write drum. (Illegal on BRLESC.)

DATA v,vl,.../c,cl,.../	Stores initial values for variables. c,cl,... represents constants.
DATA (v=c),(vl=cl),...	This is CDC form of the DATA statement.

ACKNOWLEDGEMENTS

Mr. Alfred Anderson reviewed the text and programmed the subroutine that reads and prints decimal numbers in the object programs. Messrs Michael Romanelli and George Francis also reviewed the text and offered constructive criticism.

LLOYD W. CAMPBELL

GLENN A. BECK

REFERENCES

1. Campbell, L. and Beck, G. The Instruction Code for the BRL Electronic Scientific Computer (BRLESC), Ballistic Research Laboratories Memorandum Report No. 1379, November 1961.

2. Campbell, L. and Beck, G. The FORAST Programming Language for ORDVAC and BRLESC, Ballistic Research Laboratories Report No. 1275, March 1965.

3. IBM 7090/7094 Programming Systems FORTRAN II Programming (Form C28-6054-5), 1963.

4. IBM General Information Manual, FORTRAN (Form F28-8074-1), 1961.

5. IBM Reference Manual, 709/7090 FORTRAN Operations (Form C28-6066-5), 1963.

6. IBM 7090/7094 Programming Systems FORTRAN IV Language (Form C28-6274-2), 1963.

7. FORTRAN vs Basic FORTRAN, Communications of ACM, October 1964.

8. FORTRAN 63/Reference Manual, CDC, 1964.

9. FORTRAN 66, CDC 6600 Programming System, Volume 3, 1964.

APPENDIX A

LIST OF PREDEFINED FUNCTIONS FOR BRLESC

(F indicates fl.pt. and I indicates integer)

II NAME	IV NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
ABSF	ABS	F	F	1	Absolute value.
XABSF	IABS	I	I	1	Absolute value.
INTF	AINT	F	F	1	Truncation to whole number.
XINTF	INT	F	I	1	Convert fl.pt. no. to integer.
MODF	AMOD	F	F	2	Arg.1(mod Arg.2).
XMODF	MOD	I	I	2	Arg.1(mod Arg.2).
MAXOF	AMAXO	I	F	≥ 2	Chooses largest argument.
MAX1F	AMAX1	F	F	≥ 2	Chooses largest argument.
XMAXOF	MAXO	I	I	≥ 2	Chooses largest argument.
XMAX1F	MAX1	F	I	≥ 2	Chooses largest argument.
MINOF	AMINO	I	F	≥ 2	Chooses smallest argument.
MIN1F	AMIN1	F	F	≥ 2	Chooses smallest argument.
XMINOF	MINO	I	I	≥ 2	Chooses smallest argument.
XMIN1F	MIN1	F	I	≥ 2	Chooses smallest argument.
FLOATF	FLOAT	I	F	1	Convert integer to fl.pt.
XFIXF	IFIX	F	I	1	Convert fl.pt. to integer.
SIGNF	SIGN	F	F	2	Transfer sign of Arg.2 to Arg.1.
XSIGNF	ISIGN	I	I	2	Transfer sign of Arg.2 to Arg.1.
DIMF	DIM	F	F	2	Arg.1 - minimum (Arg.1,Arg.2).
XDIMF	IDIM	I	I	2	Arg.1 - minimum (Arg.1,Arg.2)
SQRTF	SQRT	F	F	1	Square root.
SINF	SIN	F	F	1	Sine (argument in radians)
COSF	COS	F	F	1	Cosine (argument in radians)
LOGF	ALOG	F	F	1	Natural logarithm.
EXPF	EXP	F	F	1	Exponential.
ATANF	ATAN	F	F	1	Arctangent (result in radians)
TANHF	TANH	F	F	1	Hyperbolic tangent.
	ALOG10	F	F	1	Base ten logarithm.
	ATAN2	F	F	2	Arctangent of (Arg.1/Arg.2)

II NAME	IV NAME	ARGUMENT	RESULT	Number of ARGUMENTS	DEFINITION
Non-Standard functions allowed on BRLESC:					
XLOCF		F or I	I	1	Stores the address of the Arg.
	ATAN1	F	F	2	Same as ATAN2.
ARCSINF	ARCSIN	F	F	1	Arcsine.
ARCCOSF	ARCCOS	F	F	1	Arcosine.
ARCTANF	ARCTAN	F	F	1	Arctangent (same as ATANF).

APPENDIX B. THREE EXAMPLES OF FORTRAN PROGRAMS (PROGRAM, INPUT DATA, AND OUTPUT ARE LISTED)

```
*     EXAMPLE 1 MULTIPLY TWO VECTORS A*B   L.W. CAMPBELL
      DIMENSION A(10),B(10),C(10)
      READ 2,A,B
      DO 3 I=1,10
    3 C(I)=A(I)*B(I)
      PUNCH 4,C
      STOP
    4 FORMAT(14HVECTOR PRODUCT/(5E14.7))
      END
END   (THIS CARD REQUIRED ONLY ON BRLESC.)
        14.1          60.35          22.8          91.7          374.18
        36.2         193.44         83.61         2.648            9.8
        4.21           8.23          15.9          7.77           88.1
         2.            3.0         8.1118          19.1          42.44

OCT.25,63   BRLESC  FORTRAN 2
*     EXAMPLE 1 MULTIPLY TWO VECTORS A*B   L.W. CAMPBELL

VECTOR PRODUCT
 0.5936100E 02 0.4966805E 03 0.3625200E 03 0.7125090E 03 0.3296526E 05
 0.9774000E 02 0.5803200E 03 0.6782276E 03 0.5057680E 02 0.4159120E 03
```

```
*       EXAMPLE 2 FIND SMALLEST NUMBER IN ARRAY F, L.CAMPBELL
        DIMENSION F(20)
        READ 2,F
        SMALL=F(1)
        DO 9 J=2,20
        IF(SMALL-F(J))9,9,8
    8   SMALL=F(J)
    9   CONTINUE
        PUNCH 3,SMALL
        STOP
    3   FORMAT(11HSMALLEST F=F13.6)
        END
END
          14.1        60.35        22.8        91.7        374.18
          36.2       193.44        83.61       2.648         9.8
          4.21         8.23        15.9        7.77         88.1
           2.7         3.0         8.1118     19.1          42.44

OCT.25,63   BRLESC  FORTRAN 2
*       EXAMPLE 2 FIND SMALLEST NUMBER IN ARRAY F, L.CAMPBELL

SMALLEST F=      2.648000
```

```
*      EXAMPLE 22 FROM BRL REPORT 1209 CODED IN FORTRAN-L.CAMPBELL
C      USE BISECTION METHOD TO FIND ROOT OF F(X)=X**3-X-1  IN INTERVAL (1,2)
   11 FORMAT(5X,1HX10X4HF(X)//)
   21 FORMAT(1P2E15.7)
   31 FORMAT(24HCONDITIONS NOT SATISFIED)
      X=1.
      X1=2.
      EPS=.00001
      ASSIGN 1 TO K
      PUNCH 11
   44 F=X*(X*X-1.)-1.
      PUNCH21,X,F
      GOTOK,(1,4,7)
    1 F0=F
      IF(F)2,15,15
   15 XP=X
      GOTO 3
    2 XN=X
    3 X=X1
      ASSIGN 4 TO K
      GOTO 44
    4 F1=F
      IF(F)5,45,45
   45 XP=X
      GOTO 6
    5 XN=X
    6 ASSIGN 7 TO K
      IF(F0*F1)66,65,65
   65 PUNCH 31
   67 STOP
   66 X=(XN+XP)/2.
      GOTO 44
    7 IF(ABSF(F)-EPS)67,71,71
   71 IF(F)8,72,72
   72 XP=X
      GOTO 66
    8 XN=X
      GOTO 66
      END
END


OCT.25,63   BRLESC  FORTRAN 2
*      EXAMPLE 22 FROM BRL REPORT 1209 CODED IN FORTRAN-L.CAMPBELL

      X           F(X)

  1.0000000E 00 -1.0000000E 00
  2.0000000E 00  5.0000000E 00
  1.5000000E 00  8.7500000E-01
  1.2500000E 00 -2.9687500E-01
  1.3750000E 00  2.2460938E-01
  1.3125000E 00 -5.1513672E-02
  1.3437500E 00  8.2611084E-02
  1.3281250E 00  1.4575958E-02
  1.3203125E 00 -1.8710613E-02
  1.3242187E 00 -2.1279454E-03
  1.3261719E 00  6.2088296E-03
  1.3251953E 00  2.0366507E-03
  1.3247070E 00 -4.6594883E-05
  1.3249512E 00  9.9479097E-04
  1.3248291E 00  4.7403882E-04
  1.3247681E 00  2.1370716E-04
  1.3247375E 00  8.3552438E-05
  1.3247223E 00  1.8477852E-05
  1.3247147E 00 -1.4058747E-05
```

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) U.S. Army Ballistic Research Laboratories Aberdeen Proving Ground, Maryland	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b GROUP

3. REPORT TITLE

BRLESC FORTRAN IV

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(Last name, first name, initial)*

Campbell, Lloyd W. and Beck, Glenn A.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
October 1966	108	9

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. RDT&E 1P014501A14B	Report No. 1346
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	U.S. Army Materiel Command Washington, D.C.

13. ABSTRACT

FORTRAN is popular programming language that has been implemented on many computers. It is now available on Ballistic Research Laboratories' BRLESC computer. This report describes the FORTRAN language in general and includes specific details about its implementation on BRLESC.

DD FORM 1 JAN 64 1473

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Programming Language Digital Computer BRLESC Computer Compiler FORTRAN						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.